Type Inference for Rank 2 Gradual Intersection Types

Pedro Ângelo[1,2] and Mário Florido[1,3]

[1] Faculdade de Ciências & LIACC, Universidade do Porto, Porto, Portugal
[2] pedro.angelo@fc.up.pt
[3] amf@dcc.fc.up.pt

Abstract. In this paper, we extend a rank 2 intersection type system with gradual types. We then show that the problem of finding a principal typing for a lambda term, in a rank 2 gradual intersection type system is decidable. We present a type inference algorithm which builds the principal typing of a term through the generation of type constraints which are solved by a new extended unification algorithm constructing the most general unifier for rank 2 gradual intersection types.

1 Introduction

Gradual typing [5, 6, 11, 12] has earned a great deal of attention in the types research community. Aiming to seamlessly integrate static and dynamic typing, its focus is on enabling the fine-tuning of the distribution of static and dynamic type checking in a program, and to harness the strengths of both typing disciplines. The successful application [11] of gradual typing to the parametric polymorphic Hindley-Milner (HM) type system [9, 14, 20] marks an important breakthrough, showing that it is possible to apply it to statically typed functional programming languages such as Haskell or ML.

Intersection types [7, 8, 18, 25] extend the simply typed lambda-calculus [13], adding to the language of types an intersection operator $\cap$ and allowing to type terms with different types belonging to an intersection $(T_1 \cap \ldots \cap T_n)$. Intersection types provide a form of polymorphism in which it is possible to explicitly indicate every single instance of a type. Thus a term may have multiple types belonging to a finite set (intersection) of type possibilities. Although the type inference problem for intersection types is not decidable in general, it becomes decidable for finite rank fragments of the general system [17].

Recently there has been an increasing interest in intersection types for general purpose programming languages. Examples include TypeScript [26] and Flow [4]. These systems use intersection types to combine different types into one. This enables its use in contexts where the classic object-oriented model does not apply. Rank 2 intersection types [15, 16] are particularly interesting for languages with type inference: they are more powerful than parametric polymorphic types [9] for

The final authenticated version is available online at https://doi.org/10.1007/978-3-030-47147-7_5.

functional programming languages such as ML, because they type more terms, and this extra power comes for free, since the complexity of typability is identical in both systems. In fact, in the two systems typability is DEXPTIME-complete.

In this paper, we present a type inference algorithm for a rank 2 intersection gradual type system which automatically deduces the type of an expression, allowing the programmer to write code without worrying about type annotations.

If dynamic types, which are only introduced by a programmer, are allowed as instances of intersection types, expressions may be typed with both static and dynamic types simultaneously. For example, consider the following expression:

$$\lambda x : \mathit{Int} \cap \mathit{Dyn} \ . \ x \ x$$

The occurrences of the variable x may be assigned both the *Dyn* and the *Int* type. A possible assignment of types which well-types the expression is the following:

$$\lambda x : \mathit{Int} \cap \mathit{Dyn} \ . \ x^{\mathit{Dyn}} \ x^{\mathit{Int}}$$

Here we define a type inference algorithm which first generates a set of constraints on types and then solves them using an extended type unification algorithm. The first phase of type inference is to assign initial types to expressions and then generate constraints between these types. For example, consider the expression referred previously:

$$\lambda x : \mathit{Int} \cap \mathit{Dyn} \ . \ x \ x$$

Let $\dot{\lesssim}$ denote a consistent subtyping [12] constraint between two types, which means that the two types might satisfy the consistent subtyping relation. The constraint generation algorithm generates the following initial typings and corresponding constraints for the expression (several typings are generated due to different choices of where to assign types to variables):

$$\lambda x : \mathit{Int} \cap \mathit{Dyn} \ . \ x^{\mathit{Dyn}} \ x^{\mathit{Int}} : (\mathit{Int} \cap \mathit{Dyn}) \to \mathit{Dyn}$$
$$\{\mathit{Int} \ \dot{\lesssim} \ \mathit{Dyn}\}$$

$$\lambda x : \mathit{Int} \cap \mathit{Dyn} \ . \ x^{\mathit{Dyn}} \ x^{\mathit{Dyn}} : \mathit{Dyn} \to \mathit{Dyn}$$
$$\{\mathit{Dyn} \ \dot{\lesssim} \ \mathit{Dyn}\}$$

The $\dot{\lesssim}$ constraint guarantees that, when applying a function, the type of the argument is a consistent subtype of the domain type of the function. The constraint solving algorithm solves the constraints and produces a substitution of types for type variables, which when applied to the initial type assigned to the expression returns a final type for the expression. For the previous example, we will end up with the following well-typed expression as result:

$$\lambda x : \mathit{Int} \cap \mathit{Dyn} \ . \ x \ x : (\mathit{Int} \cap \mathit{Dyn} \to \mathit{Dyn}) \cap (\mathit{Dyn} \to \mathit{Dyn})$$

Thus, this paper makes the following main contributions:

1. A *type inference algorithm*: following [11], our approach first generates type constraints and then solves these constraints using a new unification algorithm for gradual intersection types of rank 2.
2. Theorems of *soundness* and *completeness* of the type inference algorithm, which show that the types returned by the algorithm are derivable in the type system and that, given an expression, the algorithm produces a syntactic description of all the types which type the expression using the type system.
3. The existence of *principal typings*, typings which represent all other typings for the same expression, for rank 2 gradual intersection types.

Related Work In [2], intersections were used to type overloaded functions which can discriminate on the type of the argument and execute different code for different types. Functions typed with intersections run different pieces of code accordingly to the type of their arguments. These systems extended semantic subtyping [10] with gradual types, and types are interpreted as sets of values. Another view of intersection types originated in the Turin group of intersection type systems [7, 8], and was also used in the programming language Forsythe [21, 22]. Intersection types are used as finitely parametric polymorphic types where functions with intersection types have a uniform behaviour: when applied to arguments of different types, they always execute the same code for all of these types. Here we follow this second approach. In previous work, we integrated gradual types with intersection types on a gradual intersection type system [29], which considered intersection types without a finite rank restriction, thus the type inference problem was not decidable. In this paper, by restricting intersection types to rank 2, we can define a type inference algorithm.

Type inference for a system with intersection and gradual types was presented before in [3]. In this contribution, constraint solving reused existing solving algorithms such as unification and tallying and, in the type inference algorithm, intersections were coded in a type language with union types, an empty type and negation types. In [3], type inference is sound but not complete, and it is semi-decidable for set-theoretical gradual types. Here we present a sound and complete type inference algorithm, where decidability is achieved by restricting the type system to types of a finite rank.

Type inference for gradual type systems is the topic of other previous works described in [24] and [11]. These systems inferred gradual types for a given expression and were also based on extended type unification algorithms which deal with type equality in the presence of dynamic types. Both systems deal with gradual types, but not intersection types. For intersection type systems, type inference [15–18] was previously defined for finite-rank intersection types, using a generalization of the unification algorithm dealing with the complicated operation of *type expansion*. These systems deal with intersection types but not gradual types.

2 Rank 2 Gradual Intersection Types

We consider a type language where intersection types are limited to rank 2, following a definition of rank 2 inspired in [16,19]. Thus, we define rank 2 gradual intersection types here:

$$\begin{aligned} T^0 &::= \; X \mid B \mid Dyn \mid T^0 \to T^0 \\ T^1 &::= \; T^0 \mid T^0 \cap \ldots \cap T^0 \\ T^2 &::= \; T^0 \mid T^1 \to T^2 \end{aligned}$$

X represents a type variable, B is the set of base types, such as Int and $Bool$, T^0 is the set of simple types, containing type variables, base types and the dynamic type and also arrow types. T^1 is the set of rank 1 types, which contain finite and non-empty intersections of simple types. Finally, T^2 represents the set of rank 2 types, which may contain intersections, but only to the left of a single arrow. We refer to the set of possible types under our system, $T^1 \cup T^2$, simply as T. The following types are considered rank 2 gradual intersection types:

$$\begin{aligned} &(T_1 \to T_1 \cap T_2 \to T_2) \to (T_1 \cap T_2) \to T \\ &((T_1 \to T_2) \cap T_1) \to T_2 \end{aligned}$$

However, these do not belong to the set of rank 2 gradual intersection types:

$$\begin{aligned} &((T_1 \to T_1) \cap (T_2 \to T_2)) \to (T_1 \cap T_2) \to (T_1 \cap T_2) \\ &((T_1 \cap T_2) \to T_1) \to T_2 \end{aligned}$$

Therefore, intersection types are not allowed in the codomain of an arrow type, agreeing with the original definition in [7]. Intersections are commutative (e.g. $T_1 \cap T_2 = T_2 \cap T_1$), idempotent (e.g. $T_1 \cap T_1 = T_1$) and associative (e.g. $(T_1 \cap T_2) \cap T_3 = T_1 \cap (T_2 \cap T_3)$. There is no distinction between a singleton intersection of types and its sole element, so for any type T, T can be considered an intersection of types of size 1. The intersection type connective $\cap$ has higher precedence (binds tighter) than the arrow type. Also, we can abbreviate an intersection type with the following definition:

$$T_1 \cap \ldots \cap T_n = \bigcap_{i=1}^{n} T_i$$

These two representations are used interchangeably.

In presenting the syntax of our language we will follow the convention that c ranges over constants such as integers and truth values, x ranges over variables, e ranges over expressions and T ranges over types. The language of expressions in our system is given by the following grammar:

$$Expressions\ e \;::=\; x \mid \lambda x : T^1 \;.\; e \mid \lambda x \;.\; e \mid e\ e \mid c$$

Note that there are two lambda abstraction expressions, one for typed code, allowing the insertion of type annotations, and another one for untyped code, which does not require type annotations. We impose one restriction on type annotations in lambda abstractions, besides being rank 1 types, they may not contain type variables X. As we are presenting a type inference algorithm, type annotations are not required since types will be inferred automatically by the algorithm. We also fix a set of term constants for the base types. For example, we might assume a base type *Int*, and the term constants are the natural numbers. In the type system, term constants have the appropriate base types. Note that if the language is only implicitly typed (without type annotations) the inferred types are static. Dynamic types are introduced only by type annotations. This design option goes back to previous work regarding type inference for gradual typing [11] where also *"there can be no dynamism without annotation"*.

A typing context is a finite set, represented by $\{x_1 : T^1_1, \ldots, x_n : T^1_n\}$, of (type variable, T^1 type) pairs called bindings. We use Γ to range over typing contexts. We write $\Gamma(x)$ for the type bounded by the variable x in the typing context Γ and define $\Gamma(x)$ as: $\Gamma(x) = T$, if $x : T \in \Gamma$. We write $dom(\Gamma)$ for the set $\{x \mid x : T \in \Gamma\}$, for all T, and $cod(\Gamma)$ for the set $\{T \mid x : T \in \Gamma\}$, for all x. We write Γ_x for the typing context Γ with any binding for the variable x removed. We define Γ_x as: $\Gamma_x = \Gamma/\{x : T\}$, for any type T.

An annotation context is a finite set, represented by $\{x_1 : T^1_1, \ldots, x_n : T^1_n\}$, of (type variable, T^1 type) pairs called bindings. We use A to range over annotation contexts. We write $A(x)$ for the type paired with the variable x in the annotation context A, defined as: $A(x) = T$ if $x : T \in A$. We write $dom(A)$ for the set $\{x \mid x : T \in A\}$, for all T. We write $cod(A)$ for the set $\{T \mid x : T \in A\}$, for all x. We write A_x for the annotation context A with any pair for the variable x removed. We define A_x as: $A_x = A/\{x : T\}$, for any type T.

3 Type System

In this section, we present the rank 2 gradual intersection type system (GITS), in Figure 1. The GITS system type checks an explicitly typed lambda-calculus language with integers and booleans. This type system is composed of type rules that originate from both gradual typing [5] and intersection types, particularly from [7]. As with gradual typing, to declare terms as either dynamically typed or statically typed, we simply add an explicit domain-type declaration in lambda abstractions.

The cornerstone of gradual typing is the $\sim$ (consistency) relation on types. We say that two types are consistent if the parts where both types are defined (static) are equal. If the expected type of an expression is an arrow type, in the T-App rule for example, but that expression is typed with the *Dyn* type, then the system assumes that the type of the expression is an arrow type. Therefore, pattern matching ($\rhd$) is a feature of gradual typing that enables the *Dyn* type to be treated as a function type from *Dyn* to *Dyn* (*Dyn* $\rightarrow$ *Dyn*), or if the type is already an arrow type, it gets its domain and codomain. Rule T-Abs: generalizes a similar rule for abstractions for the Forsythe programming language [22]. In

Syntax

$$\textit{Types } T, PM ::= B \mid Dyn \mid T \to T \mid T \cap \ldots \cap T$$
$$\textit{Expressions } e ::= x \mid \lambda x\ .\ e \mid \lambda x : T^1\ .\ e \mid e\ e \mid c$$

$\boxed{\Gamma \vdash_{\cap G} e : T}$ Typing

$$\frac{x : T_1 \cap \ldots \cap T_n \in \Gamma}{\Gamma \vdash_{\cap G} x : T_i}\ \textsc{T-Var} \qquad \frac{\Gamma, x : T_1 \vdash_{\cap G} e : T_2 \qquad static(T_1)}{\Gamma \vdash_{\cap G} \lambda x\ .\ e : T_1 \to T_2}\ \textsc{T-Abs}$$

$$\frac{\Gamma, x : T_1 \cap \ldots \cap T_m \vdash_{\cap G} e : T \qquad m \leq n}{\Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e : T_1 \cap \ldots \cap T_m \to T}\ \textsc{T-Abs:}$$

$$\frac{\begin{array}{c}\Gamma \vdash_{\cap G} e_1 : PM \qquad PM \rhd T_1 \cap \ldots \cap T_n \to T \\ \Gamma \vdash_{\cap G} e_2 : T_1' \cap \ldots \cap T_n' \qquad T_1' \lesssim T_1 \ldots T_n' \lesssim T_n\end{array}}{\Gamma \vdash_{\cap G} e_1\ e_2 : T}\ \textsc{T-App}$$

$$\frac{\Gamma \vdash_{\cap G} e : T_1\ \cdots\ \Gamma \vdash_{\cap G} e : T_n}{\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n}\ \textsc{T-Gen} \qquad \frac{\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n}{\Gamma \vdash_{\cap G} e : T_i}\ \textsc{T-Inst}$$

$$\frac{c \textit{ is a constant of type } T}{\Gamma \vdash_{\cap G} c : T}\ \textsc{T-Const}$$

$\boxed{T \rhd T}$ Pattern Matching

$$T_1 \to T_2 \rhd T_1 \to T_2 \qquad\qquad Dyn \rhd Dyn \to Dyn$$

Fig. 1. Gradual Intersection Type System ($\vdash_{\cap G}$)

this rule, the type of the formal parameter must be a subset of the set of types declared explicitly in the abstraction (as an intersection type).

We now define the subtyping ($\leq$) relation, which in this system is just a simplified version of the subtyping (or type inclusion) relation from [1]. Albeit having no use in the type system, we include subtyping in this paper because it is necessary for soundness and completeness properties. The subtyping relation is inductively defined using the following rules (bear in mind that subtyping is transitive):

Definition 1 (Subtyping).

1. $T \leq T$
2. $T_1 \cap \ldots \cap T_n \leq T_1 \cap \ldots \cap T_m$ *with* $m \leq n$
3. $T_1 \to T_2 \leq T_3 \to T_4 \iff T_3 \leq T_1 \wedge T_2 \leq T_4$
4. $T \leq T_1 \cap \ldots \cap T_n \iff T \leq T_1$ *and* $\ldots$ *and* $T \leq T_n$
5. $(T \to T_1) \cap \ldots \cap (T \to T_n) \leq T \to T_1 \cap \ldots \cap T_n$

At first glance, gradual typing and intersection types seem rather incompatible for two reasons: types in these two systems are compared using different relations, $\sim$ for gradual types and $\leq$ for intersection types; and also type inference rules for gradual typing know what type to assign a variable since only one type is annotated in abstractions while type inference rules for intersection types don't know which instance will be used for a particular occurrence of a term variable, hence assigning a type variable instead. Approaching the first incompatibility, an obvious solution would be to combine these two key relations so that they can be used in the same system while maintaining their purposes. Keeping in mind that the $\leq$ relation is not commutative, the following definition captures the essence of both relations. The consistent subtyping [12] relation is inductively defined using the following rules:

Definition 2 (Consistent Subtyping).

1. $Dyn \lesssim T$
2. $T \lesssim Dyn$
3. $T \lesssim T$
4. $T_1 \cap \ldots \cap T_n \lesssim T_1 \cap \ldots \cap T_m$ *with* $m \leq n$
5. $T_1 \to T_2 \lesssim T_3 \to T_4 \iff T_3 \lesssim T_1 \wedge T_2 \lesssim T_4$
6. $T \lesssim T_1 \cap \ldots \cap T_n \iff T \lesssim T_1 \wedge \ldots \wedge T \lesssim T_n$
7. $(T \to T_1) \cap \ldots \cap (T \to T_n) \lesssim T \to T_1 \cap \ldots \cap T_n$

In a sense, $\lesssim$ represents the $\leq$ relation from intersection types but extended to take into account the consistency of all types with the *Dyn* type, hence rules 1 and 2. Also, bear in mind that consistent subtyping is not transitive. The following cases hold under $\lesssim$:

$$Int \to Int \lesssim Int \to Int \cap Dyn$$
$$Int \to Dyn \lesssim Dyn \to Dyn$$

Now that we have overcome this first obstacle, we now define substitutions, our constraints and how they relate with substitutions.

Substitutions are the standard substitution on types but extended to deal with the *Dyn* type and intersection types. Let $[X \mapsto T^0]$ be a type substitution of X to T^0, meaning that when applied to a type T' ($[X \mapsto T^0]T'$), every occurrence of X in T' is replaced with T^0. We restrict T^0 to be a simple type, therefore, substitution cannot introduce intersection types, but only substitute type variables with simple types. A substitution applied to an intersection type is the same as applying the same substitution to each instance of the intersection type. The composition of substitutions is written as $S_1 \circ S_2$ and it is the same as applying the substitutions S_2 and then S_1, similar to the standard function composition. We sometimes write the composition of substitutions as $[X_1 \mapsto T_1, \ldots, X_n \mapsto T_n]$, which is equivalent to writing $[X_1 \mapsto T_1] \circ \ldots \circ [X_n \mapsto T_n]$. We lift substitutions to apply to expressions, by leaving the expression unchanged and substituting type annotations.

Constraints are defined by the following grammar:

$$Constraints\ C ::= T \mathrel{\dot{\lesssim}} T \mid T \doteq T \mid C \cup C$$

We define two types of constraints: the $\dot{\lesssim}$ constraint states that two types should satisfy the consistent subtyping [12] relation and the $\doteq$ constraint is the standard equality constraint. A substitution S models a constraint C ($S \models C$) between two types, T_1 and T_2, if the relation associated with that constraint holds for $S(T_1)$ and $S(T_2)$.

Definition 3 (Constraint Satisfaction).

1. $S \models \emptyset$
2. $S \models T_1 \mathrel{\dot{\lesssim}} T_2 \iff S(T_1) \lesssim S(T_2)$
3. $S \models T_1 \doteq T_2 \iff S(T_1) = S(T_2)$
4. $S \models C_1 \cup C_2 \iff S \models C_1$ *and* $S \models C_2$

The type inference algorithm will be defined bottom-up regarding the assignment of types, thus different occurrences of the same term variable may be typed with different type variables. The application of expressions containing different bindings for the same variable must join the bindings in the same typing context. The following operation combines typing contexts resulting from different derivations of the type inference algorithm. For two typing contexts Γ_1 and Γ_2, we define $\Gamma_1 + \Gamma_2$ as follows:

Definition 4 ($\Gamma_1 + \Gamma_2$). *For each* $x \in dom(\Gamma_1) \cup dom(\Gamma_2)$,

$$(\Gamma_1 + \Gamma_2)(x) = \begin{cases} \Gamma_1(x), & if\ x \notin dom(\Gamma_2) \\ \Gamma_2(x), & if\ x \notin dom(\Gamma_1) \\ \Gamma_1(x) \cap \Gamma_2(x), & otherwise \end{cases}$$

Combining typing contexts is essentially gathering the types bound to a certain variable, in multiple typing contexts, in an intersection type, for each variable in each typing context. We can abbreviate the sum of various typing contexts as following, and these two representations are used interchangeably:

$$\Gamma_1 + \ldots + \Gamma_n = \textstyle\sum_{i=1}^{n} \Gamma_i$$

4 Type Inference

Adapting ideas from the type inference algorithms for gradual typing [11] and intersection types [15], we adopt the common scheme for type inference, introduced by [27], which is to generate constraints for typeability and solve them through a constraint unification phase.

$\boxed{A \mid \Gamma \vdash_{\cap G} e : T \mid C}$ Constraint Generation

$$\frac{A(x) = T_1 \cap \ldots \cap T_n \qquad i \in 1..n \qquad \text{if } x \in dom(A)}{A \mid \{x : T_i\} \vdash_{\cap G} x : T_i \mid \{\}} \text{ C-VAR1}$$

$$\frac{X \text{ is a fresh type variable} \qquad \text{if } x \notin dom(A)}{A \mid \{x : X\} \vdash_{\cap G} x : X \mid \{\}} \text{ C-VAR2}$$

$$\frac{c \text{ is a constant of type } T}{A \mid \{\} \vdash_{\cap G} c : T \mid \{\}} \text{ C-CONST} \qquad \frac{A \mid \Gamma \vdash_{\cap G} e : T \mid C \qquad \text{if } x \in dom(\Gamma)}{A \mid \Gamma_x \vdash_{\cap G} \lambda x \,.\, e : \Gamma(x) \rightarrow T \mid C} \text{ C-ABS1}$$

$$\frac{A \mid \Gamma \vdash_{\cap G} e : T \mid C \qquad \text{if } x \notin dom(\Gamma) \qquad X \text{ is a fresh type variable}}{A \mid \Gamma \vdash_{\cap G} \lambda x \,.\, e : X \rightarrow T \mid C} \text{ C-ABS2}$$

$$\frac{A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma \vdash_{\cap G} e : T \mid C \qquad \text{if } x \in dom(\Gamma)}{A \mid \Gamma_x \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : \Gamma(x) \rightarrow T \mid C} \text{ C-ABS:1}$$

$$\frac{A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma \vdash_{\cap G} e : T \mid C \qquad \text{if } x \notin dom(\Gamma)}{A \mid \Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : (T_1 \rightarrow T) \cap \ldots \cap (T_n \rightarrow T) \mid C} \text{ C-ABS:2}$$

$$\frac{\begin{array}{c} A \mid \Gamma_1 \vdash_{\cap G} e_1 : T_1 \mid C_1 \qquad A \mid \Gamma_2 \vdash_{\cap G} e_2 : T_2 \mid C_2 \\ cod(T_1) \doteq T_3 \mid C_3 \qquad T_2 \dot{\lesssim} dom(T_1) \mid C_4 \qquad T_1 \text{ is simple type} \end{array}}{A \mid \Gamma_1 + \Gamma_2 \vdash_{\cap G} e_1 \; e_2 : T_3 \mid C_1 \cup C_2 \cup C_3 \cup C_4} \text{ C-APP}$$

$$\frac{\begin{array}{c} A \mid \Gamma \vdash_{\cap G} e_1 : T_1 \cap \ldots \cap T_n \rightarrow T \mid C \\ A \mid \Gamma_1 \vdash_{\cap G} e_2 : T'_1 \mid C_1 \;\; \ldots \;\; A \mid \Gamma_n \vdash_{\cap G} e_2 : T'_n \mid C_n \end{array}}{A \mid \Gamma + \Gamma_1 + \ldots + \Gamma_n \vdash_{\cap G} e_1 \; e_2 : T \mid C \cup C_1 \cup \{T'_1 \dot{\lesssim} T_1\} \cup \ldots \cup C_n \cup \{T'_n \dot{\lesssim} T_n\}} \text{ C-APP}\cap$$

Fig. 2. Constraint Generation

4.1 Constraint Generation

Given an annotation context A (whose elements are provided by user-supplied annotations in lambda-abstractions) and an expression e, the constraint generation algorithm $A \mid \Gamma \vdash_{\cap G} e : T \mid C$ (in Figure 2, see auxiliary definitions in Figures 3 and 4) returns a set of tuples containing a typing context Γ, a type T and a set of constraints C.

The constraint generation algorithm follows bottom-up traversing the syntactic tree of the expression. So, when assigning types to expressions, the algorithm will first assign types to the leaves of the syntactic tree of the expression, and then work its way up. This is useful for intersection types because we can assign different type variables to different instances of the same variable. This allows generating different typings for the same variable, which can be joined in the same intersection type. An issue we overcome arises from having the assignment of types working as bottom-up while also forcing certain variables to be typed with certain types, using annotations in lambda abstractions. The algorithm cannot decide which instance of the type bound by a variable in the typing

$\boxed{cod(T_1) \doteq T_2 \mid C}$

$$\frac{X_1,\ X_2 \text{ are fresh}}{cod(X) \doteq X_2 \mid \{X \doteq X_1 \rightarrow X_2\}} \qquad \frac{}{cod(T_1 \rightarrow T_2) \doteq T_2 \mid \{\}}$$

$$\frac{}{cod(Dyn) \doteq Dyn \mid \{\}}$$

Fig. 3. Constraint Codomain Judgment

$\boxed{T_2 \mathrel{\dot{\lesssim}} dom(T_1) \mid C}$

$$\frac{X_1,\ X_2 \text{ are fresh}}{T_2 \mathrel{\dot{\lesssim}} dom(X) \mid \{X \doteq X_1 \rightarrow X_2, T_2 \mathrel{\dot{\lesssim}} X_1\}} \qquad \frac{}{T_2 \mathrel{\dot{\lesssim}} dom(T_{11} \rightarrow T_{12}) \mid \{T_2 \mathrel{\dot{\lesssim}} T_{11}\}}$$

$$\frac{}{T_2 \mathrel{\dot{\lesssim}} dom(Dyn) \mid \{T_2 \mathrel{\dot{\lesssim}} Dyn\}}$$

Fig. 4. Constraint Domain judgment

context by lambda abstractions, will be assigned to a certain occurrence of that variable, before checking the context in which that variable is located. Therefore, the types of variables must be chosen before knowing how the variable's type is constrained by its use in the program.

For example, consider the following expression:

$$\lambda f\ .\ \lambda x : Int \cap Dyn\ .\ f\ (x\ x)$$

The algorithm cannot decide if it should assign type *Int* or *Dyn* to the first occurrence of variable x. According to the context, it is clear that the first occurrence should have an arrow type, which can be converted from the *Dyn* type. However, when typing x the algorithm hasn't accessed this information yet. Since in the gradual type inference defined in [11] we know what type to assign to a variable before reaching that variable, the adaptation of gradual type inference to support intersection types is not trivial. To solve this difficulty, the type inference algorithm produces various typings, each corresponding to a choice of what type to assign to that particular variable.

According to rule C-Var1, we choose an instance of the type bound by x in the annotation context A. This leads to the generation of various typings (a more complete explanation is provided in subsection 4.4). For the choices which originate an ill-typed expression, the algorithm fails, returning only the choices leading to a well-typed expression. This way we avoid committing to a single choice, which could cause a typeable expression to be rejected by the type inference. Regarding the variables x, in the previous example, the following typings are produced:

$$\{x : Int \cap Dyn\} \mid \{x : Int\} \vdash_{\cap G} x : Int \mid \{\}$$
$$\{x : Int \cap Dyn\} \mid \{x : Int\} \vdash_{\cap G} x : Int \mid \{\}$$

$$\{x : Int \cap Dyn\} \mid \{x : Int\} \vdash_{\cap G} x : Int \mid \{\}$$
$$\{x : Int \cap Dyn\} \mid \{x : Dyn\} \vdash_{\cap G} x : Dyn \mid \{\}$$

$$\{x : Int \cap Dyn\} \mid \{x : Dyn\} \vdash_{\cap G} x : Dyn \mid \{\}$$
$$\{x : Int \cap Dyn\} \mid \{x : Int\} \vdash_{\cap G} x : Int \mid \{\}$$

$$\{x : Int \cap Dyn\} \mid \{x : Dyn\} \vdash_{\cap G} x : Dyn \mid \{\}$$
$$\{x : Int \cap Dyn\} \mid \{x : Dyn\} \vdash_{\cap G} x : Dyn \mid \{\}$$

Then, by rule C-App, the algorithm checks if the type of the expression in the left-hand side is an arrow type or can be converted to one. In the first two typings, this is not true. Therefore the algorithm fails for those alternatives and proceeds for the last two alternatives.

Regarding the rules for application, the expression on the left-hand side can be typed with a type whose domain is an intersection type or a simple type. Therefore, we require two rules to discriminate between these two cases. When the domain type of the expression is a simple type, the rule for application, C-App, is the standard one from [11] with a few minor changes. Constraint Codomain Judgment (Figure 3) and the Constraint Domain Judgment (Figure 4) are adapted to deal with the $\lesssim$ relation instead of the $\sim$ relation, and thus rule C-App ensures that the type of the expression on the left-hand side of an application is an arrow type and that the domain of this arrow type is a supertype (i.e. it includes it using the subtype relation) of the type of the argument (the expression on the right-hand side of the application).

When the type of the expression on the left-hand side is an intersection type, the rule C-App$\cap$ requires the generation of different typings, one for each instance of the intersection type in the domain of the expression. Then it checks if the different types for the argument are consistent with the instances of the intersection type in the domain. This rule is inspired by an analogous rule in [15].

Both constraint generation rules will then join together the typing contexts of the two subexpressions, or in the case of rule C-App$\cap$, the typing contexts of the different typings, by combining the types bound to the same variables as an intersection type, according to Definition 4.

The next lemmas show that the constraint generation algorithm is both sound and complete, w.r.t. the type system.

Lemma 1 (Constraint Soundness). *If $A \mid \Gamma \vdash_{\cap G} e : T \mid C$ and $S \models C$ then $S(\Gamma) \vdash_{\cap G} S(e) : S(T)$.*

Proof. By induction on the length of the derivation tree of $A \mid \Gamma \vdash_{\cap G} e : T \mid C$.

Lemma 2 (Constraint Completeness). *If* $\Gamma_1 \vdash_{\cap G} e : T_1$ *then*

1. *there exists a derivation* $A \mid \Gamma_2 \vdash_{\cap G} e : T_2 \mid C$ *such that* $\exists S \,.\, S \models C$
2. *for* $A \mid \Gamma_{21} \vdash_{\cap G} e : T_{21} \mid C_1$ *such that* $\exists S_1 \,.\, S_1 \models C_1$ *and* $\ldots$ *and* $A \mid \Gamma_{2n} \vdash_{\cap G} e : T_{2n} \mid C_n$ *such that* $\exists S_n \,.\, S_n \models C_n$ *then*
   (a) *for each* $x \in dom(\Gamma_1) \cap dom(\sum_{i=1}^{n} \Gamma_{2i})$, $\Gamma_1(x) \leq S_i(\Gamma_{2i}(x))$, $\forall i \in 1..n$
   (b) $\bigcap_{i=1}^{n} S_i(T_{2i}) \leq T_1$

Proof. By induction on the length of the derivation tree of $\Gamma_1 \vdash_{\cap G} e : T_1$.

4.2 Constraint Solving

Given a set of constraints C, obtained by constraint generation, we shall define, in Figure 5, a solving relation between a set of constraints C and a substitution S ($C \Rightarrow S$) meaning: solving the set of constraints C results in S. Rules in Figure 5 are syntax-directed and define a decision algorithm by successively applying these rules using a bottom-up proof search strategy.

Our constraint solving algorithm extends Robinson unification [23] to deal with new equality definitions which account for dynamic types and intersection types. Most of these rules are adapted from [5] and [15], with a few exceptions. Since there are two types of constraints, there are two groups of constraint solving rules, and also a base case to halt the algorithm (rule Em). The constraint solving algorithm first transforms any $\dot{\lesssim}$ constraint into an equivalent standard unification problem involving only equality constraints. Thus, there is an order of application of rules in the constraint solver defined in Figure 5. First, rules CS transform $\dot{\lesssim}$ constraints into a set of equations. Then, rules Eq, solve the resulting set of equations yielding a substitution as the solution for the initial set of constraints.

Given that $\dot{\lesssim}$ constraints are a new concept, a brief walkthrough of the rules will clarify their meaning. Most rules that deal with $\dot{\lesssim}$ are a direct adaptation of [15] and relate to subtyping (definition 1). Only rules CS-DynL and CS-DynR stand out, since they are used to simulate $\sim$ from [11]. The remaining rules, which regard $\doteq$ constraints, come from [11]. When we have a $\dot{\lesssim}$ constraint between different type variables or base types, we constrain those types to be equal, since they cannot be solved further. The remaining rules, for the $\doteq$ constraint, are based on standard unification rules for equality.

Going back to the example above, the two alternatives that haven't failed, produce the following typings and constraints:

$$\lambda f \,.\, \lambda x : Int \cap Dyn \,.\, f \ (x \ x) : X_1 \rightarrow (Int \cap Dyn) \rightarrow X_3$$
$$\{Int \dot{\lesssim} Dyn, X_1 \doteq X_2 \rightarrow X_3, X_1 \doteq X_4 \rightarrow X_5, Dyn \dot{\lesssim} X_4\}$$

$$\lambda f \,.\, \lambda x : Int \cap Dyn \,.\, f \ (x \ x) : X_1 \rightarrow Dyn \rightarrow X_3$$
$$\{Dyn \dot{\lesssim} Dyn, X_1 \doteq X_2 \rightarrow X_3, X_1 \doteq X_4 \rightarrow X_5, Dyn \dot{\lesssim} X_4\}$$

$\boxed{C \Rightarrow S}$ Constraint Solving

$$\frac{}{\emptyset \Rightarrow \emptyset}\ \textsc{Em} \qquad \frac{C \Rightarrow S}{\{Dyn \mathrel{\dot{\lesssim}} T\} \cup C \Rightarrow S}\ \text{CS-}\textsc{DynL} \qquad \frac{C \Rightarrow S}{\{T \mathrel{\dot{\lesssim}} Dyn\} \cup C \Rightarrow S}\ \text{CS-}\textsc{DynR}$$

$$\frac{C \Rightarrow S \qquad T \in \{Int, Bool\} \cup TVar}{\{T \mathrel{\dot{\lesssim}} T\} \cup C \Rightarrow S}\ \text{CS-}\textsc{Refl} \qquad \frac{C \Rightarrow S \qquad m \leq n}{\{T_1 \cap \ldots \cap T_n \mathrel{\dot{\lesssim}} T_1 \cap \ldots \cap T_m\} \cup C \Rightarrow S}\ \text{CS-}\textsc{Inst}$$

$$\frac{C \Rightarrow S}{\{(T \to T_1) \cap \ldots \cap (T \to T_n) \mathrel{\dot{\lesssim}} T \to T_1 \cap \ldots \cap T_n\} \cup C \Rightarrow S}\ \text{CS-}\textsc{Assoc}$$

$$\frac{\{T_3 \mathrel{\dot{\lesssim}} T_1, T_2 \mathrel{\dot{\lesssim}} T_4\} \cup C \Rightarrow S}{\{T_1 \to T_2 \mathrel{\dot{\lesssim}} T_3 \to T_4\} \cup C \Rightarrow S}\ \text{CS-}\textsc{Arrow} \qquad \frac{\{T \mathrel{\dot{\lesssim}} T_1, \ldots, T \mathrel{\dot{\lesssim}} T_n\} \cup C \Rightarrow S}{\{T \mathrel{\dot{\lesssim}} T_1 \cap \ldots \cap T_n\} \cup C \Rightarrow S}\ \text{CS-}\textsc{InstR}$$

$$\frac{\begin{array}{c}\{X_1 \mathrel{\dot{\lesssim}} T_1, T_2 \mathrel{\dot{\lesssim}} X_2, T \doteq X_1 \to X_2\} \cup C \Rightarrow S \\ X_1,\ X_2 \text{ are fresh type variables}\end{array}}{\{T_1 \to T_2 \mathrel{\dot{\lesssim}} T\} \cup C \Rightarrow S}\ \text{CS-}\textsc{ArrowL}$$

$$\frac{\begin{array}{c}\{T_1 \mathrel{\dot{\lesssim}} X_1, X_2 \mathrel{\dot{\lesssim}} T_2, T \doteq X_1 \to X_2\} \cup C \Rightarrow S \\ X_1,\ X_2 \text{ are fresh type variables}\end{array}}{\{T \mathrel{\dot{\lesssim}} T_1 \to T_2\} \cup C \Rightarrow S}\ \text{CS-}\textsc{ArrowR}$$

$$\frac{\{T_1 \doteq T_2\} \cup C \Rightarrow S \qquad T_1, T_2 \in \{Int, Bool\} \cup TVar}{\{T_1 \mathrel{\dot{\lesssim}} T_2\} \cup C \Rightarrow S}\ \text{CS-}\textsc{Eq}$$

$$\frac{C \Rightarrow S \qquad T \in \{Int, Bool\} \cup TVar}{\{T \doteq T\} \cup C \Rightarrow S}\ \textsc{Eq-Refl} \qquad \frac{\{T_1 \doteq T_3, T_2 \doteq T_4\} \cup C \Rightarrow S}{\{T_1 \to T_2 \doteq T_3 \to T_4\} \cup C \Rightarrow S}\ \textsc{Eq-Arrow}$$

$$\frac{\{X \doteq T\} \cup C \Rightarrow S \qquad T \notin TVar}{\{T \doteq X\} \cup C \Rightarrow S}\ \textsc{Eq-VarR} \qquad \frac{[X \mapsto T]C \Rightarrow S \qquad X \notin Vars(T)}{\{X \doteq T\} \cup C \Rightarrow S \circ [X \mapsto T]}\ \textsc{Eq-VarL}$$

Fig. 5. Constraint Solving

Since the step by step solving of the constraints produced for each typing are equal, only one solving will be shown. Applying the first step (rule CS-DynR) leads both constraint sets to:

$$\{X_1 \doteq X_2 \to X_3, X_1 \doteq X_4 \to X_5, Dyn \mathrel{\dot{\lesssim}} X_4\}$$

By rule Eq-VarL, the constraint set is reduced to

$$\{X_2 \to X_3 \doteq X_4 \to X_5, Dyn \mathrel{\dot{\lesssim}} X_4\}$$

and the first substitution is produced: $[X_1 \mapsto X_2 \to X_3]$. Then, by rule Eq-Arrow, the constraint set is further reduced to

$$\{X_2 \doteq X_4, X_3 \doteq X_5, Dyn \mathrel{\dot{\lesssim}} X_4\}$$

Applying rule Eq-VarL two times reduces the constraint set to just one constraint

$$\{Dyn \mathrel{\dot{\lesssim}} X_4\}$$

and updates the substitutions to $[X_3 \mapsto X_5, X_2 \mapsto X_4, X_1 \mapsto X_2 \to X_3]$. Finally, solving the remaining constraint gives as final the substitutions:

$$[X_3 \mapsto X_5, X_2 \mapsto X_4, X_1 \mapsto X_2 \to X_3]$$

The final typings of the expressions are then:

$$\lambda f\ .\ \lambda x : \mathit{Int} \cap \mathit{Dyn}\ .\ f\ (x\ x) : (X_4 \to X_5) \to (\mathit{Int} \cap \mathit{Dyn} \to X_5)$$
$$\lambda f\ .\ \lambda x : \mathit{Int} \cap \mathit{Dyn}\ .\ f\ (x\ x) : (X_4 \to X_5) \to (\mathit{Dyn} \to X_5)$$

This extended unification algorithm used for constraint solving is both sound and complete, with respect to constraint satisfaction (definition 3). Note that completeness means that the extended unification algorithm produces most general unifiers.

Lemma 3 (Unification Soundness). *If $C \Rightarrow S$ then $S \models C$.*

Proof. By induction on the length of the derivation tree of $C \Rightarrow S$.

Lemma 4 (Unification Completeness). *If $S_1 \models C$ then $C \Rightarrow S_2$ for some S_2, and furthermore $S_1 = S \circ S_2$ for some S.*

Proof. We proceed by induction on the breakdown of constraint sets by the unification rules.

4.3 Gradual Types

Any type is a consistent subtype, or consistent supertype, of the *Dyn* type, thus there is no need for further checks, such as recursively checking consistent subtyping through the structure of the type. Constraints which require a type to be consistent subtype, or supertype, with the *Dyn* type have been discarded up until now using our definition of constraint solving since they are satisfiable with any substitution. Discarding these constraints brings a problem regarding the instantiation of type variables. A type variable that is only constrained to be consistent with the *Dyn* type will not be substituted since no substitution concerning that variable will be produced. However, as that type variable is only constrained by the *Dyn* type, it should be instantiated to the *Dyn* type, so a substitution from that variable to the *Dyn* type should be produced. Implementing this only takes a simple extension [11] to our constraint solving algorithm. Therefore, given a set of constraints C, the constraint solving algorithm $G \mid C \Rightarrow S$ will produce a set of substitutions S and a set of gradual types G. The extension is shown in Figure 6.

To instantiate these unconstrained type variables to *Dyn*, we first need to collect them. When any constraint of the form $T \mathrel{\dot{\lesssim}} \mathit{Dyn}$ or $\mathit{Dyn} \mathrel{\dot{\lesssim}} T$ is encountered by the solver, we store the type T, per rules CS-DynL and CS-DynR. Note that these types might be constrained by other constraints, however, we collect them

$\boxed{G \mid C \Rightarrow S}$ Constraint Unification

$$\frac{}{G \mid \emptyset \Rightarrow \overline{[Vars(G) \mapsto Dyn]}}\ \textsc{Em} \qquad \frac{G \cup \{T\} \mid C \Rightarrow S}{G \mid \{Dyn \overset{\cdot}{\lesssim} T\} \cup C \Rightarrow S}\ \text{CS-}\textsc{DynL}$$

$$\frac{G \cup \{T\} \mid C \Rightarrow S}{G \mid \{T \overset{\cdot}{\lesssim} Dyn\} \cup C \Rightarrow S}\ \text{CS-}\textsc{DynR}$$

$$\frac{[X \mapsto T]G \mid [X \mapsto T]C \Rightarrow S \qquad X \notin Vars(T)}{G \mid \{X \doteq T\} \cup C \Rightarrow S \circ [X \mapsto T]}\ \textsc{Eq-VarL}$$

Fig. 6. Constraint Solving with Gradual Types

nonetheless. These will be considered gradual types since they potentially contain the *Dyn* type. When a constraint is solved and a substitution is produced, the constraint solver applies the substitution to the remaining constraints to avoid unconstrained type variables. This behaviour must also be implemented, regarding the gradual types stored. In rule Eq-VarL, when a substitution is produced, it is applied to the remaining constraints and also to the collection of gradual types. Finally, when all constraints have been solved and all the substitutions have been produced, we will get the complete collection of gradual types. These will possibly contain base types, such as *Int*, compound types such as the arrow type and type variables. Then, we take the type variables from these types and produce substitutions from those type variables to *Dyn*. This is done by rule Em. $Vars(G)$ is the set of all the type variables present in all the types in G. The overline means that a substitution will be produced for each type variable obtained by $Vars(G)$.

Since the constraint unification algorithm has been updated, we need to update the soundness and completeness lemmas to match the new algorithm's specification.

Lemma 5 (Unification Soundness). *If $G \mid C \Rightarrow S$ then $S \models C$.*

Proof. Extends proof of Lemma 3. By induction on the length of the derivation tree of $G \mid C \Rightarrow S$.

Lemma 6 (Unification Completeness). *If $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} \models C$ then $G \mid C \Rightarrow S_2$ for some S_2, and furthermore $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} = S \circ S_2$ for some S.*

Proof. Extends proof of Lemma 4. By induction on the breakdown of constraint sets by the unification rules.

Continuing the example above, with the extended constraint solving algorithm, a final substitution is added:

$$[X_4 \mapsto Dyn, X_3 \mapsto X_5, X_2 \mapsto X_4, X_1 \mapsto X_2 \to X_3]$$

The final typings of the expressions are then:

$$\lambda f\ .\ \lambda x : Int \cap Dyn\ .\ f\ (x\ x) : (Dyn \to X_5) \to (Int \cap Dyn \to X_5)$$
$$\lambda f\ .\ \lambda x : Int \cap Dyn\ .\ f\ (x\ x) : (Dyn \to X_5) \to (Dyn \to X_5)$$

Notice that only in the first solution all the instances of the type in the annotation of the lambda abstraction are used.

4.4 Multiple Solutions

In the language described in Section 2, variables may be annotated with intersection types in lambda abstractions. In these cases, the type inference algorithm assigns a particular instance of that intersection type to a particular occurrence of that variable. However, given the fact that we are dealing with idempotent intersection types, we cannot know in advance which instance to assign to a particular occurrence of a variable since some choices lead to ill-typed expressions while other choices lead to well-typed expressions. For example, consider the following expression,

$$\lambda x : Int \cap Dyn\ .\ x\ x\ x$$

We must choose, for each of the three occurrences of x, either the *Int* or the *Dyn* type. Some choices lead to the expression becoming ill-typed, such as:

$$\lambda x : Int \cap Dyn\ .\ x^{Int}\ x^{Dyn}\ x^{Int}$$

Other choices lead the expression to become well-typed, such as:

$$\lambda x : Int \cap Dyn\ .\ x^{Dyn}\ x^{Dyn}\ x^{Int}$$
$$\lambda x : Int \cap Dyn\ .\ x^{Dyn}\ x^{Int}\ x^{Int}$$

Therefore, our type inference algorithm first produces several typings for an expression. Since there are many different choices to type variables, we generate different typings according to each choice. The generation of multiple typings is clear in rule C-Var1, which generates a typing for a variable for each instance of intersection type bound to that variable in the annotation context.

Constraint generation produces several sets of constraints and each set of constraints is solved by the constraint solving algorithm leading to multiple incomparable solutions. We will show that the type inference algorithm is sound and complete and that the set of substitutions computed by the algorithm is principal in the sense that any other solution is an instance of one in the set returned by the solver when it is applied to the different constraint sets produced in the constraint generation phase.

The expression $\lambda x : Int \cap Dyn\ .\ x\ x\ x$ has a total of 8 typings, which correspond to choosing different combinations of *Int* and *Dyn* for the three occurrences of the variable x. We can see that of those choices, only 4 will produce a typeable expression. Choosing *Int* for the first occurrence of x leads to an ill-typed expression. Therefore, we end up with 4 different typings:

$$\lambda x : Int \cap Dyn\ .\ x^{Dyn}\ x^{Int}\ x^{Int} : Int \cap Dyn \to Dyn$$
$$\lambda x : Int \cap Dyn\ .\ x^{Dyn}\ x^{Dyn}\ x^{Int} : Int \cap Dyn \to Dyn$$
$$\lambda x : Int \cap Dyn\ .\ x^{Dyn}\ x^{Int}\ x^{Dyn} : Int \cap Dyn \to Dyn$$
$$\lambda x : Int \cap Dyn\ .\ x^{Dyn}\ x^{Dyn}\ x^{Dyn} : Dyn \to Dyn$$

However, note that the last typing does not use all the instances in typing variables. The type inference algorithm is then described as follows:

Definition 5 (Type Inference). *Let e be an expression, Γ a context, T a type, S a substitution and Sol a set of triples of the form (Γ, T, S). The type inference function I from expressions to sets of triples (Γ, T, S), is defined by the following steps:*

1. *$Sol = \emptyset$*
2. *for every derivation of $\emptyset \mid \Gamma \vdash_{\cap G} e : T \mid C$ that holds*
   (a) if $\emptyset \mid C \Rightarrow S$ holds then
   $Sol = Sol \cup \{(S(\Gamma), S(T), S)\}$
3. *return Sol*

Step 2 generates constraints with derivations in the constraint generation system. Given an empty annotation context and the expression e, $\emptyset \mid \Gamma \vdash_{\cap G} e : T \mid C$ gets us the typing context Γ, the type of the expression T and the set of constraints C. In step 2.a, given an empty set of gradual types and the constraints C, if the constraint solver algorithm $\emptyset \mid C \Rightarrow S$ produces a substitution S, then that substitutions S is added to the solutions.

4.5 Decidability

Different typings in the constraint generation system in Figure 2 arise from intersections, and intersections are always finite, thus the number of derivations for a given expression is also finite. Also, since constraint generation follows the syntactic tree of the expression, each constraint generation derivation terminates.

Lemma 7 (Termination of Constraint Generation). *Given a context A and an expression e, the number of derivations by the constraint generation system for $A \mid \Gamma \vdash_{\cap G} e : T \mid C$ is finite.*

Proof. The proof follows by structural induction on e.

Now, to prove that the successive application of constraint solving rules in Figure 5 always halt, note that, every rule, when applied to a consistent subtyping constraint, reduces the number of type constructors in consistent subtyping constraints or reduces the number of consistent subtyping constraints. If the rule applies to an equality constraint then every rule reduces the number of type constructors in equality constraints or reduces the number of equality constraints. The only rule that has a different behaviour is EQ-VARR, but it will be followed by rule EQ-VARL which reduces the number of equality constraints. Thus to prove termination we use a metric well-ordered by a lexicographical order on the tuples $(\mathrm{NICS}, \mathrm{NCCS}, \mathrm{NCS})$ and $(\mathrm{NVE}\textsc{q}, \mathrm{NCE}\textsc{q}, \mathrm{NTXE}\textsc{q}, \mathrm{NE}\textsc{q})$, where NICS is the number of unique intersection types in the left of an $\dot{\lesssim}$ constraint + the number of unique intersection types in the right of an $\dot{\lesssim}$ constraint; NCCS is the number of type constructors in $\dot{\lesssim}$ constraints; NCS is the number of $\dot{\lesssim}$ constraints; NVEQ is the number of different type variables in $\doteq$ constraints; NCEQ is the number of type constructors in $\doteq$ constraints; NTXEQ is the number of $\doteq$ constraints of the form $T \doteq X$; and NEQ is the number of $\doteq$ constraints. The result is stated in the following lemma.

Lemma 8 (Termination of Constraint Solving). *$C \Rightarrow S$ terminates for every set of constraints C.*

Proof. By a metric well-ordered by a lexicographical order. The full proof can be consulted in Appendix A.

Finally, decidability of the type inference algorithm follows from the two last lemmas.

Theorem 1 (Decidability). *Type inference is decidable.*

Proof. By lemmas 7 and 8

4.6 Soundness and Completeness

Soundness and completeness are two important properties which show the correctness and usefulness of the type inference algorithm. Soundness guarantees that if the type inference algorithm returns a type, then that type is derivable in the type system. Completeness states that the output of the type inference algorithm represents the *most general type judgment able to type the expression*, a property known as *principal typing*. The full proofs of the following theorems can be consulted in Appendix A.

Theorem 2 (Soundness). *If $(\Gamma, T, S) \in I(e)$ then $S(\Gamma) \vdash_{\cap G} S(e) : S(T)$.*

Proof. By lemmas 1 and 5.

Principal Typing A type judgment, or typing, for a term, is *principal if and only if all other typings for the same expression can be derived from it by some set of operations*. Thus principal typings can be seen as the most general typings. The notion of principal typing and its relation with the slightly different notion of principal type was studied in detail in [16, 28].

Definition 6 (Principal Typing). *If $\Gamma_p \vdash_{\cap G} e : T_p$, then we say that (Γ_p, T_p) is a principal typing of e if whenever $\Gamma_1 \vdash_{\cap G} e : T_1$ holds, then for some substitutions S, for each $x \in dom(\Gamma_1) \cap dom(\Gamma_p)$, we have $\Gamma_1(x) \leq S(\Gamma_p(x))$ and $S(T_p) \leq T_1$.*

As the following theorem shows, our language has principal typings for every well-typed expression.

Theorem 3 (Principal Typings). *If $\Gamma_1 \vdash_{\cap G} e : T_1$ then there are $\Gamma_{21}, \ldots, \Gamma_{2n}$, $T_{21}, \ldots, T_{2n}$, $S_{21}, \ldots, S_{2n}$ and $S_1, \ldots, S_n$ such that $((\Gamma_{21}, T_{21}, S_{21}), \ldots, (\Gamma_{2n}, T_{2n}, S_{2n})) = I(e)$ and, for each $x \in dom(\Gamma_1) \cap dom(\Gamma_{21} + \ldots + \Gamma_{2n})$, we have $\Gamma_1(x) \leq S_1 \circ S_{21}(\Gamma_{21}(x))$ and $\ldots$ and $\Gamma_1(x) \leq S_n \circ S_{2n}(\Gamma_{2n}(x))$ and $S_1 \circ S_{21}(T_{21}) \cap \ldots \cap S_n \circ S_{2n}(T_{2n}) \leq T_1$.*

Proof. By lemmas 2 and 6.

Principal typings are clearly a quite relevant feature of our type system. They allow compositional type inference, where type inference for a given expression uses only the typings inferred for its subexpressions, which can be inferred independently in any order.

5 Conclusion

Here we study the type inference problem for the rank 2 fragment of our general system and prove that it is decidable, by defining a type inference algorithm, sound w.r.t. the type system and complete in the sense that returns principal typings. This strongly indicates that rank 2 intersection gradual types may be safely and successfully applied to the design and implementation of gradually typed programming languages able to type values which are all of many different types.

Acknowledgments

This work is partially funded by FCT within project Elven POCI-01-0145-FEDER-016844, Project 9471 - Reforçar a Investigação, o Desenvolvimento Tecnológico e a Inovação (Project 9471-RIDTI) and by Fundo Comunitário Europeu FEDER.

References

1. Henk Barendregt, Mario Coppo, and Mariangiola Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *Journal of Symbolic Logic*, 48(4):931–940, 1983.
2. Giuseppe Castagna and Victor Lanvin. Gradual typing with union and intersection types. *Proc. ACM Program. Lang.*, 1(ICFP):41:1–41:28, August 2017.
3. Giuseppe Castagna, Victor Lanvin, Tommaso Petrucciani, and Jeremy G. Siek. Gradual typing: A new perspective. *Proc. ACM Program. Lang.*, 3(POPL):16:1–16:32, January 2019.
4. Avik Chaudhuri. Flow: Abstract interpretation of javascript for type checking and beyond. In *Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, PLAS '16. ACM, 2016.
5. Matteo Cimini and Jeremy G. Siek. The gradualizer: A methodology and algorithm for generating gradual type systems. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 443–455, 2016.
6. Matteo Cimini and Jeremy G. Siek. Automatically generating the dynamic semantics of gradually typed languages. In *Proceedings of the 44th ACM SIGPLAN Symposium on Principles of Programming Languages*, POPL 2017, pages 789–803, 2017.
7. M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre Dame Journal of Formal Logic*, 21(4):685–693, 1980.
8. M. Coppo, M. Dezani-Ciancaglini, and B. Venneri. Functional characters of solvable terms. *Mathematical Logic Quarterly*, 27(2-6):45–58, 1981.
9. Luis Damas and Robin Milner. Principal type-schemes for functional programs. In *Proceedings of the 9th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '82, pages 207–212, 1982.
10. Alain Frisch, Giuseppe Castagna, and Véronique Benzaken. Semantic subtyping: Dealing set-theoretically with function, union, intersection, and negation types. *J. ACM*, 55(4):19:1–19:64, September 2008.
11. Ronald Garcia and Matteo Cimini. Principal type schemes for gradual programs. In *Proceedings of the 42nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '15, pages 303–315, 2015.
12. Ronald Garcia, Alison M. Clark, and Éric Tanter. Abstracting gradual typing. In *Proceedings of the 43rd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '16, pages 429–442, 2016.
13. J. Roger Hindley. *Basic Simple Type Theory*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1997.
14. R. Hindley. The principal type-scheme of an object in combinatory logic. *Transactions of the American Mathematical Society*, 146:29–60, 1969.
15. T. Jim. Rank 2 type systems and recursive definitions. Technical report, Cambridge, MA, USA, 1995.
16. Trevor Jim. What are principal typings and what are they good for? In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '96, pages 42–53, 1996.
17. A. J. Kfoury and J. B. Wells. Principality and decidable type inference for finite-rank intersection types. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, pages 161–174, 1999.

18. A.J. Kfoury and J.B. Wells. Principality and type inference for intersection types using expansion variables. *Theoretical Computer Science*, 311(1):1 – 70, 2004.
19. Daniel Leivant. Polymorphic type inference. In *Proceedings of the 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '83, pages 88–98, 1983.
20. Robin Milner. A theory of type polymorphism in programming. *Journal of Computer and System Sciences*, 17(3):348 – 375, 1978.
21. John C. Reynolds. The coherence of languages with intersection types. In *Proceedings of the International Conference on Theoretical Aspects of Computer Software*, TACS '91, pages 675–700. Springer-Verlag, 1991.
22. John C. Reynolds. *Design of the Programming Language Forsythe*, pages 173–233. Birkhäuser Boston, Boston, MA, 1997.
23. J. A. Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1):23–41, January 1965.
24. Jeremy G. Siek and Manish Vachharajani. Gradual typing with unification-based inference. In *Proceedings of the 2008 Symposium on Dynamic Languages*, DLS '08, pages 7:1–7:12, 2008.
25. Steffen van Bakel. Intersection type assignment systems. *Theoretical Computer Science*, 151(2):385 – 435, 1995.
26. Panagiotis Vekris, Benjamin Cosman, and Ranjit Jhala. Refinement types for typescript. *SIGPLAN Not.*, 51(6):310–325, June 2016.
27. Mitchell Wand. A simple algorithm and proof for type inference. *Fundamenta Informaticae*, 10(2):115–121, 1987.
28. J. B. Wells. The essence of principal typings. In *Automata, Languages and Programming, 29th International Colloquium, ICALP 2002, Malaga, Spain, July 8-13, 2002, Proceedings*, pages 913–925, 2002.
29. Pedro Ângelo and Mário Florido. Gradual intersection types. In *Ninth Workshop on Intersection Types and Related Systems, ITRS 2018, Oxford, U.K., 8 July 2018*, 2018.

A Proofs

Lemma 9 (Weakening). *If $\Gamma \vdash_{\cap G} e : T$ then $\Gamma + \Gamma' \vdash_{\cap G} e : T$ for any typing context Γ'.*

Proof. We proceed by induction on the derivation tree of $\Gamma \vdash_{\cap G} e : T$.

Base cases:

- Rule T-Var. If $\Gamma \vdash_{\cap G} x : T_i$ then $x : T_1 \cap \ldots \cap T_n \in \Gamma$. If $x : T'_1 \cap \ldots \cap T'_m \in \Gamma'$, then $x : T_1 \cap \ldots \cap T_n \cap T'_1 \cap \ldots \cap T'_m \in \Gamma + \Gamma'$. Therefore, $\Gamma + \Gamma' \vdash_{\cap G} x : T_i$.
- Rule T-Const. If $\Gamma \vdash_{\cap G} c : T$ and c is a constant of type T, then $\Gamma + \Gamma' \vdash_{\cap G} c : T$.

Induction step:

- Rule T-Abs. To avoid capture we assume that $\alpha - reduction$ is made whenever needed to rename formal parameters. If $\Gamma \vdash_{\cap G} \lambda x \,.\, e : T_1 \to T_2$ then $\Gamma, x : T_1 \vdash_{\cap G} e : T_2$. By induction hypothesis, $\Gamma, x : T_1 + \Gamma' \vdash_{\cap G} e : T_2$. By rule T-Abs, $\Gamma + \Gamma' \vdash_{\cap G} \lambda x \,.\, e : T_1 \to T_2$.
- Rule T-Abs:. To avoid capture we assume that $\alpha - reduction$ is made whenever needed to rename formal parameters. If $\Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : T_1 \cap \ldots \cap T_m \to T$ then $\Gamma, x : T_1 \cap \ldots \cap T_m \vdash_{\cap G} e : T$. By induction hypothesis, $\Gamma, x : T_1 \cap \ldots \cap T_m + \Gamma' \vdash_{\cap G} e : T$. By rule T-Abs:, $\Gamma + \Gamma' \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : T_1 \cap \ldots \cap T_m \to T$.
- Rule T-App. If $\Gamma \vdash_{\cap G} e_1\ e_2 : T$ then $\Gamma \vdash_{\cap G} e_1 : PM, PM \rhd T_1 \cap \ldots \cap T_n \to T$, $\Gamma \vdash_{\cap G} e_2 : T'_1 \cap \ldots \cap T'_n$ and $T'_1 \lesssim T_1 \ldots T'_n \lesssim T_n$. By induction hypothesis, $\Gamma + \Gamma' \vdash_{\cap G} e_1 : PM$ and $\Gamma + \Gamma' \vdash_{\cap G} e_2 : T'_1 \cap \ldots \cap T'_n$. By rule T-App, $\Gamma + \Gamma' \vdash_{\cap G} e_1\ e_2 : T$.
- Rule T-Gen. If $\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$ then $\Gamma \vdash_{\cap G} e : T_1$ and $\ldots$ and $\Gamma \vdash_{\cap G} e : T_n$. By induction hypothesis, $\Gamma + \Gamma' \vdash_{\cap G} e : T_1$ and $\ldots$ and $\Gamma + \Gamma' \vdash_{\cap G} e : T_n$. By rule T-Gen, $\Gamma + \Gamma' \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$.
- Rule T-Inst. If $\Gamma \vdash_{\cap G} e : T_i$ then $\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$. By induction hypothesis, $\Gamma + \Gamma' \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$. By rule T-Inst, $\Gamma + \Gamma' \vdash_{\cap G} e : T_i$

Lemma 1 (Constraint Soundness). *If $A \mid \Gamma \vdash_{\cap G} e : T \mid C$ and $S \models C$ then $S(\Gamma) \vdash_{\cap G} S(e) : S(T)$.*

Proof. We proceed by induction on the length of the derivation tree of $A \mid \Gamma \vdash_{\cap G} e : T \mid C$.

Base cases:

- Rule C-Var1. If $A \mid \{x : T_i\} \vdash_{\cap G} x : T_i \mid \{\}$ and $S \models \{\}$ then $\{x : S(T_i)\} \vdash_{\cap G} x : S(T_i)$. Since $S(\{x : T_i\}) = \{x : S(T_i)\}$ and $S(x) = x$, then $S(\{x : T_i\}) \vdash_{\cap G} S(x) : S(T_i)$.
- Rule C-Var2. If $A \mid \{x : X\} \vdash_{\cap G} x : X \mid \{\}$ and $S \models \{\}$ then $\{x : S(X)\} \vdash_{\cap G} x : S(X)$. Since $S(\{x : X\}) = \{x : S(X)\}$ and $S(x) = x$, then $S(\{x : X\}) \vdash_{\cap G} S(x) : S(X)$.

– Rule C-Const. If $A \mid \{\} \vdash_{\cap G} c : T \mid \{\}$ and $S \models \emptyset$ then c is a constant of type T. Therefore, $S(\{\}) \vdash_{\cap G} S(c) : S(T)$.

Induction step:

– Rule C-Abs1. If $A \mid \Gamma_x \vdash_{\cap G} \lambda x\ .\ e : \Gamma(x) \rightarrow T \mid C$ and $S \models C$ then $A \mid \Gamma \vdash_{\cap G} e : T \mid C$. By the induction hypothesis, $S(\Gamma) \vdash_{\cap G} S(e) : S(T)$. Then, by rule T-Abs, $S(\Gamma)_x \vdash_{\cap G} \lambda x\ .\ S(e) : S(\Gamma(x)) \rightarrow S(T)$. As $S(\Gamma_x) = S(\Gamma)_x$, $S(\lambda x\ .\ e) = \lambda x\ .\ S(e)$ and $S(\Gamma(x) \rightarrow T) = S(\Gamma(x)) \rightarrow S(T)$ then $S(\Gamma_x) \vdash_{\cap G} S(\lambda x\ .\ e) : S(\Gamma(x) \rightarrow T)$.
– Rule C-Abs2. If $A \mid \Gamma \vdash_{\cap G} \lambda x\ .\ e : X \rightarrow T \mid C$ and $S \models C$ then $A \mid \Gamma \vdash_{\cap G} e : T \mid C$. By the induction hypothesis, $S(\Gamma) \vdash_{\cap G} S(e) : S(T)$. As $x : S(X)$ is not used to type e and thus $x \notin \Gamma$ then we also have $S(\Gamma) \cup \{x : S(X)\} \vdash_{\cap G} S(e) : S(T)$. Then by the T-Abs, $S(\Gamma) \vdash_{\cap G} S(\lambda x\ .\ e) : S(X \rightarrow T)$.
– Rule C-Abs:1. If $A \mid \Gamma_x \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e : \Gamma(x) \rightarrow T \mid C$ and $S \models C$ then $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma \vdash_{\cap G} e : T \mid C$. By the induction hypothesis, $S(\Gamma) \vdash_{\cap G} S(e) : S(T)$. Therefore, $S(\Gamma)_x \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n\ .\ S(e) : S(\Gamma(x)) \rightarrow S(T)$. As $S(\Gamma_x) = S(\Gamma)_x$, $S(\Gamma(x) \rightarrow T) = S(\Gamma(x)) \rightarrow S(T)$ and $\{x : T_1 \cap \ldots \cap T_m\} \in \Gamma$ then $S(\Gamma_x) \vdash_{\cap G} \lambda x : S(T_1 \cap \ldots \cap T_m) \cap T_{m+1} \cap \ldots \cap T_n\ .\ S(e) : S(\Gamma(x) \rightarrow T)$. As $T_{m+1} \cap \ldots \cap T_n$ does not occur in e, then those those types are not affected by substitutions. Therefore, $S(\Gamma_x) \vdash_{\cap G} S(\lambda x : T_1 \cap \ldots \cap T_n\ .\ e) : S(\Gamma(x) \rightarrow T)$.
– Rule C-Abs:2. If $A \mid \Gamma \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n\ .\ e : T_1 \rightarrow T \cap \ldots \cap T_n \rightarrow T \mid C$ and $S \models C$ then $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma \vdash_{\cap G} e : T \mid C$. By the induction hypothesis, $S(\Gamma) \vdash_{\cap G} S(e) : S(T)$. As $x \notin dom(\Gamma)$ then x doesn't occur in e. Therefore, we also have $S(\Gamma) \cup \{x : S(T_1)\} \vdash_{\cap G} S(e) : S(T)$ and … and $S(\Gamma) \cup \{x : S(T_n)\} \vdash_{\cap G} S(e) : S(T)$. Then, by rule T-Abs:, $S(\Gamma) \vdash_{\cap G} S(\lambda x : T_1 \cap \ldots \cap T_n\ .\ e) : S(T_1 \rightarrow T)$ and … and $S(\Gamma) \vdash_{\cap G} S(\lambda x : T_1 \cap \ldots \cap T_n\ .\ e) : S(T_n \rightarrow T)$. By rule T-Gen, we have $S(\Gamma) \vdash_{\cap G} S(\lambda x : T_1 \cap \ldots \cap T_n\ .\ e) : S(T_1 \rightarrow T \cap \ldots \cap T_n \rightarrow T)$.
– Rule C-App. If $A \mid \Gamma_1 + \Gamma_2 \vdash_{\cap G} e_1\ e_2 : T_3 \mid C_1 \cup C_2 \cup C_3 \cup C_4$ and $S \models C_1 \cup C_2 \cup C_3 \cup C_4$ then $A \mid \Gamma_1 \vdash_{\cap G} e_1 : T_1 \mid C_1$ and $A \mid \Gamma_2 \vdash_{\cap G} e_2 : T_2 \mid C_2$ and $cod(T_1) \doteq T_3 \mid C_3$ and $T_2 \dot{\lesssim} dom(T_1) \mid C_4$. There are three possibilities:
  - $T_1 = X$. Then, $T_3 = X_2$. By the induction hypothesis, $S(\Gamma_1) \vdash_{\cap G} S(e_1) : S(X)$ and $S(\Gamma_2) \vdash_{\cap G} S(e_2) : S(T_2)$. As $S \models \{X \doteq X_1 \rightarrow X_2, X \doteq X_3 \rightarrow X_4, T_2 \dot{\lesssim} X_1\}$, then $S(\Gamma_1) \vdash_{\cap G} S(e_1) : S(X_1 \rightarrow X_2)$ and $S(T_2) \lesssim S(X_1)$. Therefore, $S(\Gamma_1) \vdash_{\cap G} S(e_1) : S(X_1) \rightarrow S(X_2)$. Therefore, by Lemma 9, $S(\Gamma_1 + \Gamma_2) \vdash_{\cap G} S(e_1\ e_2) : S(X_2)$.
  - $T_1 = T_{11} \rightarrow T_{12}$. Then, $T_3 = T_{12}$. By the induction hypothesis, $S(\Gamma_1) \vdash_{\cap G} S(e_1) : S(T_{11} \rightarrow T_{12})$ and $S(\Gamma_2) \vdash_{\cap G} S(e_2) : S(T_2)$. Therefore, $S(\Gamma_1) \vdash_{\cap G} S(e_1) : S(T_{11}) \rightarrow S(T_{12})$. As $S \models T_2 \dot{\lesssim} T_{11}$, then $S(T_2) \lesssim S(T_{11})$. Therefore, by Lemma 9, $S(\Gamma_1 + \Gamma_2) \vdash_{\cap G} S(e_1\ e_2) : S(T_{12})$.
  - $T_1 = Dyn$. Then $T_3 = Dyn$. By the induction hypothesis, $S(\Gamma_1) \vdash_{\cap G} S(e_1) : S(Dyn)$ and $S(\Gamma_2) \vdash_{\cap G} S(e_2) : S(T_2)$. Therefore, $S(\Gamma_1) \vdash_{\cap G} S(e_1) : Dyn$ and $Dyn \rhd Dyn \rightarrow Dyn$. As $S(T_2) \lesssim Dyn$ then, by Lemma 9, $S(\Gamma_1 + \Gamma_2) \vdash_{\cap G} S(e_1\ e_2) : S(Dyn)$.

– Rule C-App$\cap$. If $A \mid \Gamma + \Gamma_1 + \ldots + \Gamma_n \vdash_{\cap G} e_1\ e_2 : T \mid C \cup C_1 \cup \{T_1' \dot{\lesssim} T_1\} \cup \ldots \cup C_n \cup \{T_n' \dot{\lesssim} T_n\}$ and $S \models C \cup C_1 \cup \{T_1' \dot{\lesssim} T_1\} \cup \ldots \cup C_n \cup \{T_n' \dot{\lesssim} T_n\}$ then $A \mid \Gamma \vdash_{\cap G} e_1 : T_1 \cap \ldots \cap T_n \to T \mid C$ and $A \mid \Gamma_1 \vdash_{\cap G} e_2 : T_1' \mid C_1$ and $\ldots$ and $A \mid \Gamma_n \vdash_{\cap G} e_2 : T_n' \mid C_n$ and $S(T_1') \lesssim S(T_1)$ and $\ldots$ and $S(T_n') \lesssim S(T_n)$. By the induction hypothesis, $S(\Gamma) \vdash_{\cap G} S(e_1) : S(T_1 \cap \ldots \cap T_n \to T)$ and $S(\Gamma_1) \vdash_{\cap G} S(e_2) : S(T_1')$ and $\ldots$ and $S(\Gamma_n) \vdash_{\cap G} S(e_2) : S(T_n')$. Since, by Lemma 9, $S(\Gamma + \Gamma_1 + \ldots + \Gamma_n) \vdash_{\cap G} S(e_1) : S(T_1 \cap \ldots \cap T_n) \to S(T)$, $S(\Gamma + \Gamma_1 + \ldots + \Gamma_n) \vdash_{\cap G} S(e_2) : S(T_1')$ and $\ldots$ and $S(\Gamma + \Gamma_1 + \ldots + \Gamma_n) \vdash_{\cap G} S(e_2) : S(T_n')$, then by rule T-App, $S(\Gamma + \Gamma_1 + \ldots + \Gamma_n) \vdash_{\cap G} S(e_1\ e_2) : S(T)$.

Lemma 10 (Consistent Subtyping to Subtyping). *If $T_1 \lesssim T_2$ and both T_1 and T_2 are static, then $T_1 \leq T_2$.*

Proof. We proceed by induction on definition 2.

Base cases:

– $T \lesssim T$. If $T \lesssim T$ then $T \leq T$.
– $T_1 \cap \ldots \cap T_n \lesssim T_1$ and $\ldots$ and $T_1 \cap \ldots \cap T_n \lesssim T_n$. If $T_1 \cap \ldots \cap T_n \lesssim T_1$ and $\ldots$ and $T_1 \cap \ldots \cap T_n \lesssim T_n$, then $T_1 \cap \ldots \cap T_n \leq T_1$ and $\ldots$ and $T_1 \cap \ldots \cap T_n \leq T_n$.
– $(T \to T_1) \cap \ldots \cap (T \to T_n) \lesssim T \to T_1 \cap \ldots \cap T_n$. If $(T \to T_1) \cap \ldots \cap (T \to T_n) \lesssim T \to T_1 \cap \ldots \cap T_n$ then $(T \to T_1) \cap \ldots \cap (T \to T_n) \leq T \to T_1 \cap \ldots \cap T_n$.

Induction step:

– $T_1 \to T_2 \lesssim T_3 \to T_4 \iff T_3 \lesssim T_1 \wedge T_2 \lesssim T_4$. There are two possibilities:
  - We proceed first for the right direction of the implication. If $T_1 \to T_2 \lesssim T_3 \to T_4$ then $T_3 \lesssim T_1$ and $T_2 \lesssim T_4$. By the induction hypothesis, $T_3 \leq T_1$ and $T_2 \leq T_4$. Then by the Definition 1, $T_1 \to T_2 \leq T_3 \to T_4$.
  - We now proceed for the left direction of the implication. If $T_3 \lesssim T_1$ and $T_2 \lesssim T_4$ then $T_1 \to T_2 \lesssim T_3 \to T_4$. By the induction hypothesis, $T_1 \to T_2 \leq T_3 \to T_4$. By Definition 1, $T_3 \leq T_1$ and $T_2 \leq T_4$.
– $T \lesssim T_1 \cap \ldots \cap T_n \iff T \lesssim T_1 \wedge \ldots \wedge T \lesssim T_n$. There are two possibilities:
  - We proceed first for the right direction of the implication. If $T \lesssim T_1 \cap \ldots \cap T_n$ then $T \lesssim T_1$ and $\ldots$ and $T \lesssim T_n$. By the induction hypothesis, $T \leq T_1$ and $\ldots$ and $T \leq T_n$. Therefore, by Definition 1, $T \leq T_1 \cap \ldots \cap T_n$.
  - We now proceed for the left direction of intersection types. If $T \lesssim T_1$ and $\ldots$ and $T \lesssim T_n$ then $T \lesssim T_1 \cap \ldots \cap T_n$. By the induction hypothesis, $T \leq T_1 \cap \ldots \cap T_n$. By Definition 1, $T \leq T_1$ and $\ldots$ and $T \leq T_n$.

Lemma 2 (Constraint Completeness). *If $\Gamma_1 \vdash_{\cap G} e : T_1$ then*

1. *there exists a derivation $A \mid \Gamma_2 \vdash_{\cap G} e : T_2 \mid C$ such that $\exists S\ .\ S \models C$*
2. *for $A \mid \Gamma_{21} \vdash_{\cap G} e : T_{21} \mid C_1$ such that $\exists S_1\ .\ S_1 \models C_1$ and $\ldots$ and $A \mid \Gamma_{2n} \vdash_{\cap G} e : T_{2n} \mid C_n$ such that $\exists S_n\ .\ S_n \models C_n$ then*
   (a) for each $x \in dom(\Gamma_1) \cap dom(\sum_{i=1}^{n} \Gamma_{2i})$, $\Gamma_1(x) \leq S_i(\Gamma_{2i}(x))$, $\forall i \in 1..n$
   (b) $\bigcap_{i=1}^{n} S_i(T_{2i}) \leq T_1$

Proof. We proceed by induction on the length of the derivation tree of $\mathit{\Gamma}_1 \vdash_{\cap G} e : T_1$.

Base cases:

- Rule T-Var. If $\mathit{\Gamma}_1 \vdash_{\cap G} x : T_i$ then $x : T_1 \cap \ldots \cap T_n \in \mathit{\Gamma}_1$. There are two possibilities:
  - $x \in dom(A)$. If $x \in dom(A)$, then $x : T_1 \cap \ldots \cap T_n \in A$, since the type $T_1 \cap \ldots \cap T_n$ came from the annotation of the lambda abstraction that binds x. To prove 1., we have that $A \mid \{x : T_1\} \vdash_{\cap G} x : T_1 \mid \emptyset$ and for a $S_1 = []$, $S_1 \models \emptyset$ and $\ldots$ and $A \mid \{x : T_n\} \vdash_{\cap G} x : T_n \mid \emptyset$ and for a $S_n = []$, $S_n \models \emptyset$. To prove 2.a), we have that since $S_1(\mathit{\Gamma}_{21}(x)) = T_1$ and $\ldots$ and $S_n(\mathit{\Gamma}_{2n}(x)) = T_n$ and $\mathit{\Gamma}_1(x) = T_1 \cap \ldots \cap T_n$ then by Definition 1, $\mathit{\Gamma}_1(x) \leq S_1(\mathit{\Gamma}_{21}(x))$ and $\ldots$ and $\mathit{\Gamma}_1(x) \leq S_n(\mathit{\Gamma}_{2n}(x))$ and to prove 2.b), we have that $S_1(T_1) \cap \ldots \cap S_n(T_n) \leq T_i$.
  - $x \notin dom(A)$. To prove 1., we have that $A \mid \{x : X_1\} \vdash_{\cap G} x : X_1 \mid \emptyset$ and for a $S_1 = [X_1 \mapsto T_1]$, $S_1 \models \emptyset$ and $\ldots$ and $A \mid \{x : X_n\} \vdash_{\cap G} x : X_n \mid \emptyset$ and for a $S_n = [X_n \mapsto T_n]$, $S_n \models \emptyset$. To prove 2.a), since $S_1(\mathit{\Gamma}_{21}(x)) = T_1$ and $\ldots$ and $S_n(\mathit{\Gamma}_{2n}(x)) = T_n$ and $\mathit{\Gamma}_1(x) = T_1 \cap \ldots \cap T_n$ then by Definition 1, $\mathit{\Gamma}_1(x) \leq S_1(\mathit{\Gamma}_{21}(x))$ and $\ldots$ and $\mathit{\Gamma}_1(x) \leq S_n(\mathit{\Gamma}_{2n}(x))$ and to prove 2.b), we have that $S_1(X_1) \cap \ldots \cap S_n(X_n) \leq T_i$.
- Rule T-Const. If $\mathit{\Gamma} \vdash_{\cap G} c : T$, then c is an constant of type T. Therefore, to prove 1., we have that $A \mid \{\} \vdash_{\cap G} c : T \mid \{\}$ and $S \models \emptyset$. Since there is no $x \in dom(\mathit{\Gamma}_1) \cap dom(\{\})$, 2.a) is proved. To prove 2.b), we have that $S(T) \leq T$, by Definition 1.

Induction step:

- Rule T-Abs. If $\mathit{\Gamma}_1 \vdash_{\cap G} \lambda x\ .\ e : T_1 \to T_2$ then $\mathit{\Gamma}_1, x : T_1 \vdash_{\cap G} e : T_2$. There are two possibilities:
  - $x \in dom(\mathit{\Gamma}_2)$. By the induction hypothesis on 1., exists $A \mid \mathit{\Gamma}_2 \vdash_{\cap G} e : T_2' \mid C$ such that $\exists S\ .\ S \models C$.

    By the induction hypothesis on 2., we have that for $A \mid \mathit{\Gamma}_{21} \vdash_{\cap G} e : T_{21} \mid C_1$ such that $\exists S_1\ .\ S_1 \models C_1$ and $\ldots$ and for $A \mid \mathit{\Gamma}_{2n} \vdash_{\cap G} e : T_{2n} \mid C_n$ such that $\exists S_n\ .\ S_n \models C_n$, then for each $y \in dom(\mathit{\Gamma}_1, x : T_1) \cap dom(\sum_{i=1}^{n} \mathit{\Gamma}_{2i})$, we have $(\mathit{\Gamma}_1, x : T_1)(y) \leq S_i(\mathit{\Gamma}_{2i}(y))$, $\forall i \in 1..n$, and $\bigcap_{i=1}^{n} S_i(T_{2i}) \leq T_2$.

    To prove 1., we have that as $A \mid \mathit{\Gamma}_2 \vdash_{\cap G} e : T_2' \mid C$ such that $\exists S\ .\ S \models C$, then by rule C-Abs1, exists $A \mid \mathit{\Gamma}_{2x} \vdash_{\cap G} \lambda x\ .\ e : \mathit{\Gamma}_2(x) \to T_2' \mid C$ and $S \models C$.

    To prove 2., we have that for $A \mid \mathit{\Gamma}_{21} \vdash_{\cap G} e : T_{21} \mid C_1$ then $A \mid \mathit{\Gamma}_{21x} \vdash_{\cap G} \lambda x\ .\ e : \mathit{\Gamma}_{21}(x) \to T_{21} \mid C_1$ and $S_1 \models C_1$ and $\ldots$ and for $A \mid \mathit{\Gamma}_{2n} \vdash_{\cap G} e : T_{2n} \mid C_n$ then $A \mid \mathit{\Gamma}_{2nx} \vdash_{\cap G} \lambda x\ .\ e : \mathit{\Gamma}_{2n}(x) \to T_{2n} \mid C_n$ and $S_n \models C_n$.

To prove 2.a), as $(\Gamma_1, x : T_1)(y) \leq S_1(\Gamma_{21}(y))$ and $\ldots$ and $(\Gamma_1, x : T_1)(y) \leq S_n(\Gamma_{2n}(y))$ for each $y \in dom(\Gamma_1, x : T_1) \cap dom(\Gamma_2)$ then $(\Gamma_1)(y) \leq S_1(\Gamma_{21x}(y))$ and $\ldots$ and $(\Gamma_1)(y) \leq S_n(\Gamma_{2nx}(y))$.

To prove 2.b), as $S_1(T_{21}) \cap \ldots \cap S_n(T_{2n}) \leq T_2$ and $T_1 \leq S_1(\Gamma_{21}(x))$ and $\ldots$ and $T_1 \leq S_n(\Gamma_{2n}(x))$ then by Definition 1, rule 4, $T_1 \leq S_1(\Gamma_{21}(x)) \cap \ldots \cap S_n(\Gamma_{2n}(x))$. Therefore, by Definition 1, rule 3, $S_1(\Gamma_{21}(x)) \cap \ldots \cap S_n(\Gamma_{2n}(x)) \to S_1(T_{21}) \cap \ldots \cap S_n(T_{2n}) \leq T_1 \to T_2$. Therefore, by Definition 1, rule 5, $(S_1(\Gamma_{21}(x)) \cap \ldots \cap S_n(\Gamma_{2n}(x)) \to S_1(T_{21})) \cap \ldots \cap (S_1(\Gamma_{21}(x)) \cap \ldots \cap S_n(\Gamma_{2n}(x)) \to S_n(T_{2n})) \leq T_1 \to T_2$. By Definition 1, rule 2, $S_1(\Gamma_{21}(x) \to T_{21}) \cap \ldots \cap S_n(\Gamma_{2n}(x) \to T_{2n}) \leq T_1 \to T_2$.

- $x \notin dom(\Gamma_2)$. By the induction hypothesis on 1., exists $A \mid \Gamma_2 \vdash_{\cap G} e : T_2' \mid C$ such that $\exists S \,.\, S \models C$.

  By the induction hypothesis on 2., we have that for $A \mid \Gamma_{21} \vdash_{\cap G} e : T_{21} \mid C_1$ such that $\exists S_1 \,.\, S_1 \models C_1$ and $\ldots$ and for $A \mid \Gamma_{2n} \vdash_{\cap G} e : T_{2n} \mid C_n$ such that $\exists S_n \,.\, S_n \models C_n$ then for each $y \in dom(\Gamma_1, x : T_1) \cap dom(\sum_{i=1}^{n} \Gamma_{2i})$, we have $(\Gamma_1, x : T_1)(y) \leq S_i(\Gamma_{2i}(y))$, $\forall i \in 1..n$ and $\bigcap i = 1^n S_i(T_{2i}) \leq T_2$.

  To prove 1., we have that as $A \mid \Gamma_2 \vdash_{\cap G} e : T_2' \mid C$ such that $\exists S \,.\, S \models C$ then by rule C-Abs2, exists $A \mid \Gamma_2 \vdash_{\cap G} \lambda x \,.\, e : X \to T_2' \mid C$ and $S \models C$.

  To prove 2., we have that for $A \mid \Gamma_{21} \vdash_{\cap G} e : T_{21} \mid C_1$ then $A \mid \Gamma_{21} \vdash_{\cap G} \lambda x \,.\, e : X_1 \to T_{21} \mid C_1$ and $S_1 \models C_1$ and $\ldots$ and for $A \mid \Gamma_{2n} \vdash_{\cap G} e : T_{2n} \mid C_n$ then $A \mid \Gamma_{2n} \vdash_{\cap G} \lambda x \,.\, e : X_n \to T_{2n} \mid C_n$ and $S_n \models C_n$.

  Since X_1 is a fresh type variable, it is not contained in C_1 and $\ldots$ and since X_n is a fresh type variable, it is not contained in C_n. Then, we can consider $S_1 = S_1' \circ [X_1 \mapsto T_1]$ and $\ldots$ and we can consider $S_n = S_n' \circ [X_n \mapsto T_1]$.

  To prove 2.a), as for each $y \in dom(\Gamma_1, x : T_1) \cap dom(\sum_{i=1}^{n} \Gamma_{2i})$, we have $(\Gamma_1, x : T_1)(y) \leq S_i(\Gamma_{2i}(y))$, $\forall i \in 1..n$, then $\Gamma_1(y) \leq S_i(\Gamma_{2ix}(y))$, $\forall i \in 1..n$.

  To prove 2.b), as $T_1 \leq S_1(X_1)$ and $\ldots$ and $T_1 \leq S_n(X_n)$ then by Definition 1, rule 4, $T_1 \leq S_1(X_1) \cap \ldots \cap S_n(X_n)$. As $S_1(T_{21}) \cap \ldots \cap S_n(T_{2n}) \leq T_2$, then by Definition 1, rule 3, $S_1(X_1) \cap \ldots \cap S_n(X_n) \to S_1(T_{21}) \cap \ldots \cap S_n(T_{2n}) \leq T_1 \to T_2$. Therefore, by Definition 1, rule 5, $(S_1(X_1) \cap \ldots \cap S_n(X_n) \to S_1(T_{21})) \cap \ldots \cap (S_1(X_1) \cap \ldots \cap S_n(X_n) \to S_n(T_{2n})) \leq T_1 \to T_2$. By Definition 1, rule 2, $S_1(X_1 \to T_{21}) \cap \ldots \cap S_n(X_n \to T_{2n}) \leq T_1 \to T_2$.

– Rule T-Abs:. If $\Gamma_1 \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : T_1 \cap \ldots \cap T_m \to T$ then $\Gamma_1, x : T_1 \cap \ldots \cap T_m \vdash_{\cap G} e : T$. There are two possibilities:
  - $x \in dom(\Gamma_2)$. By the induction hypothesis on 1., exists $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma_2 \vdash_{\cap G} e : T' \mid C$ such that $\exists S \,.\, S \models C$.

By the induction hypothesis on 2., we have that for $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma_{21} \vdash_{\cap G} e : T'_1 \mid C_1$ such that $\exists S_1 \,.\, S_1 \models C_1$ and ... and for $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma_{2l} \vdash_{\cap G} e : T'_l \mid C_l$ such that $\exists S_l \,.\, S_l \models C_l$ then for each $y \in dom(\Gamma_1, x : T_1 \cap \ldots \cap T_m) \cap dom(\sum_{i=1}^{l} \Gamma_{2i})$, we have that $(\Gamma_1, x : T_1 \cap \ldots \cap T_m)(y) \leq S_i(\Gamma_{2i}(y))$, $\forall i \in 1..l$, and $\bigcap_{i=1}^{l} S_i(T'_i) \leq T$.

To prove 1., we have that as $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma_2 \vdash_{\cap G} e : T' \mid C$ such that $\exists S \,.\, S \models C$, then $A \mid \Gamma_{2x} \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : \Gamma_2(x) \to T' \mid C$ and $S \models C$.

To prove 2., we have that for $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma_{21} \vdash_{\cap G} e : T'_1 \mid C_1$ then $A \mid \Gamma_{21x} \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : \Gamma_{21}(x) \to T'_1 \mid C_1$ and $S_1 \models C_1$ and ... and for $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma_{2l} \vdash_{\cap G} e : T'_l \mid C_l$ then $A \mid \Gamma_{2lx} \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : \Gamma_{2l}(x) \to T'_l \mid C_l$ and $S_l \models C_l$.

To prove 2.a), as for each $y \in dom(\Gamma_1) \cap dom(\sum_{i=1}^{l} \Gamma_{2i})$, we have $(\Gamma_1, x : T_1 \cap \ldots \cap T_m)(y) \leq S_i(\Gamma_{2i}(y))$, $\forall i \in 1..l$, then $\Gamma_1(y) \leq S_i(\Gamma_{2ix}(y))$.

To prove 2.b), we have that $T_1 \cap \ldots \cap T_m \leq S_1(\Gamma_{21}(x))$ and ... and $T_1 \cap \ldots \cap T_m \leq S_l(\Gamma_{2l}(x))$. As $T_1 \cap \ldots \cap T_m \leq S_1(\Gamma_{21}(x))$ and ... and $T_1 \cap \ldots \cap T_m \leq S_l(\Gamma_{2l}(x))$ then by Definition 1, rule 4, $T_1 \cap \ldots \cap T_m \leq S_1(\Gamma_{21}(x)) \cap \ldots \cap S_l(\Gamma_{2l}(x))$. As $S_1(T'_1) \cap \ldots \cap S_l(T'_l) \leq T$, then by Definition 1, rule 3, $S_1(\Gamma_{21}(x)) \cap \ldots \cap S_l(\Gamma_{2l}(x)) \to S_1(T'_1) \cap \ldots \cap S_l(T'_l) \leq T_1 \cap \ldots \cap T_m \to T$. Therefore, by Definition 1, rule 5, $(S_1(\Gamma_{21}(x)) \cap \ldots \cap S_l(\Gamma_{2l}(x)) \to S_1(T'_1)) \cap \ldots \cap (S_1(\Gamma_{21}(x)) \cap \ldots \cap S_l(\Gamma_{2l}(x)) \to S_l(T'_l)) \leq T_1 \cap \ldots \cap T_m \to T$. By Definition 1, rule 2, $S_1(\Gamma_{21}(x) \to T'_1) \cap \ldots \cap S_l(\Gamma_{2l}(x) \to T'_l) \leq T_1 \cap \ldots \cap T_m \to T$.

- $x \notin dom(\Gamma_2)$. By the induction hypothesis on 1., exists $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma_2 \vdash_{\cap G} e : T' \mid C$ such that $\exists S \,.\, S \models C$.

  By the induction hypothesis on 2., we have that for $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma_{21} \vdash_{\cap G} e : T'_1 \mid C_1$ such that $\exists S_1 \,.\, S_1 \models C_1$ and ... and for $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma_{2l} \vdash_{\cap G} e : T'_l \mid C_l$ such that $\exists S_l \,.\, S_l \models C_l$ then for each $y \in dom(\Gamma_1, x : T_1 \cap \ldots \cap T_m) \cap dom(\sum_{i=1}^{l} \Gamma_{2i})$, we have that $(\Gamma_1, x : T_1 \cap \ldots \cap T_m)(y) \leq S_i(\Gamma_{2i}(y))$, $\forall i \in 1..l$, and $\bigcap_{i=1}^{l} S_i(T'_i) \leq T$.

  To prove 1., we have that as $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma_2 \vdash_{\cap G} e : T' \mid C$ such that $\exists S \,.\, S \models C$ then by rule C-Abs:2, exists $A \mid \Gamma_2 \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : T_1 \to T' \cap \ldots \cap T_n \to T' \mid C$ and $S \models C$.

  To prove 2., we have that for $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma_{21} \vdash_{\cap G} e : T'_1 \mid C_1$ then $A \mid \Gamma_{21} \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : T_1 \to T'_1 \cap \ldots \cap T_n \to T'_1 \mid C_1$ and $S_1 \models C_1$ and ... and for $A_x \cup \{x : T_1 \cap \ldots \cap T_n\} \mid \Gamma_{2l} \vdash_{\cap G} e : T'_l \mid C_l$ then $A \mid \Gamma_{2l} \vdash_{\cap G} \lambda x : T_1 \cap \ldots \cap T_n \,.\, e : T_1 \to T'_l \cap \ldots \cap T_n \to T'_l \mid C_l$ and

$S_n \models C_n$.

To prove 2.a), as for each $y \in dom(\Gamma_1, x : T_1 \cap \ldots \cap T_m) \cap dom(\sum_{i=1}^{l} \Gamma_{2i})$, we have that $(\Gamma_1, x : T_1 \cap \ldots \cap T_m)(y) \leq S_i(\Gamma_{2i}(y))$, $\forall i \in 1..l$, then $\Gamma_1(y) \leq S_i(\Gamma_{2i}(y))$.

To prove 2.b), as x does not occur in e, then T_1 and $\ldots$ and T_n are not affected by $S_1, \ldots, S_n$. Therefore $S_1(T_1 \cap \ldots \cap T_n) = T_1 \cap \ldots \cap T_n$ and $\ldots$ and $S_l(T_1 \cap \ldots \cap T_n) = T_1 \cap \ldots \cap T_n$. Therefore, $S_1((T_1 \to T'_1) \cap \ldots \cap (T_n \to T'_1)) \cap \ldots \cap S_l((T_1 \to T'_l) \cap \ldots \cap (T_n \to T'_l)) = (T_1 \to S_1(T'_1)) \cap \ldots \cap (T_n \to S_1(T'_1)) \cap \ldots \cap (T_1 \to S_l(T'_l)) \cap \ldots \cap (T_n \to S_l(T'_l))$. Then, by Definition 1, rule 2, $(T_1 \to S_1(T'_1)) \cap \ldots \cap (T_n \to S_1(T'_1)) \cap \ldots \cap (T_1 \to S_l(T'_l)) \cap \ldots \cap (T_n \to S_l(T'_l)) \leq (T_1 \cap \ldots \cap T_m \to S_1(T'_1)) \cap \ldots \cap (T_1 \cap \ldots \cap T_m \to S_l(T'_l))$. Then, by Definition 1, rule 5, $(T_1 \cap \ldots \cap T_m \to S_1(T'_1)) \cap \ldots \cap (T_1 \cap \ldots \cap T_m \to S_l(T'_l)) \leq T_1 \cap \ldots \cap T_m \to S_1(T'_1) \cap \ldots \cap S_l(T'_l)$. Then, by Definition 1, rule 3, $T_1 \cap \ldots \cap T_m \to S_1(T'_1) \cap \ldots \cap S_l(T'_l) \leq T_1 \cap \ldots \cap T_m \to T$.

– Rule T-App. If $\Gamma \vdash_{\cap G} e_1\ e_2 : T$ then $\Gamma \vdash_{\cap G} e_1 : PM$, $PM \rhd T_1 \cap \ldots \cap T_n \to T$, $\Gamma \vdash_{\cap G} e_2 : T'_1 \cap \ldots \cap T'_n$ and $T'_1 \lesssim T_1$ and $\ldots$ and $T'_n \lesssim T_n$. There are two possibilities:
  - Using rule C-App. By the induction hypothesis on 1., exists $A \mid \Gamma_1 \vdash_{\cap G} e_1 : PM' \mid C_1$ such that $\exists S_1 \,.\, S_1 \models C_1$ and exists $A \mid \Gamma_2 \vdash_{\cap G} e_2 : T'' \mid C_2$ such that $\exists S_2 \,.\, S_2 \models C_2$.

    By the induction hypothesis on 2., we have that for $A \mid \Gamma_{11} \vdash_{\cap G} e_1 : PM_1 \mid C_{11}$ such that $\exists S_{11} \,.\, S_{11} \models C_{11}$ and $\ldots$ and $A \mid \Gamma_{1n'} \vdash_{\cap G} e_1 : PM_{1n'} \mid C_{1n'}$ such that $\exists S_{1n'} \,.\, S_{1n'} \models C_{1n'}$ then for each $x \in dom(\Gamma) \cap dom(\sum_{i=1}^{n'} \Gamma_{1i})$, we have that $\Gamma(x) \leq S_{1i}(\Gamma_{1i}(x))$ and $\bigcap_{i=1}^{n'} S_{1i}(PM_i) \leq PM$.

    Also, by the induction hypothesis on 2., we have that for $A \mid \Gamma_{21} \vdash_{\cap G} e_2 : T''_1 \mid C_{21}$ such that $\exists S_{21} \,.\, S_{21} \models C_{21}$ and $\ldots$ and $A \mid \Gamma_{2m'} \vdash_{\cap G} e_2 : T''_{m'} \mid C_{2m'}$ such that $\exists S_{2m'} \,.\, S_{2m'} \models C_{2m'}$ then for each $x \in dom(\Gamma) \cap dom(\sum_{j=1}^{m'} \Gamma_{2j})$, we have that $\Gamma(x) \leq S_{2j}(\Gamma_{2j}(x))$ and $\bigcap_{j=1}^{m'} S_{2j}(T''_j) \leq T'_1 \cap \ldots \cap T'_n$.

    To prove 1., we want to prove that since $A \mid \Gamma_1 \vdash_{\cap G} e_1 : PM' \mid C_1$ such that $\exists S_1 \,.\, S_1 \models C_1$ and since $A \mid \Gamma_2 \vdash_{\cap G} e_2 : T'' \mid C_2$ such that $\exists S_2 \,.\, S_2 \models C_2$, and for $cod(PM') \doteq T_3 \mid C_3$ and $T'' \mathrel{\dot{\lesssim}} dom(PM') \mid C_4$, then exists $A \mid \Gamma_1 + \Gamma_2 \vdash_{\cap G} e_1\ e_2 : T_3 \mid C_1 \cup C_2 \cup C_3 \cup C_4$ such that $\exists S_k \,.\, S_k \models C_1 \cup C_2 \cup C_3 \cup C_4$.

    To prove 2., we want to prove that, for $\forall i \in 1..n'$ and $\forall j \in 1..m'$ such that $A \mid \Gamma_{1i} \vdash_{\cap G} e_1 : PM_i \mid C_{1i}$ such that $\exists S_{1i} \,.\, S_{1i} \models C_{1i}$, $A \mid \Gamma_{2j} \vdash_{\cap G} e_2 : T''_j \mid C_{2j}$ such that $\exists S_{2j} \,.\, S_{j2} \models C_{2j}$, $cod(PM_i) \doteq T_{3i} \mid C_{3i}$ and $T''_j \mathrel{\dot{\lesssim}} dom(PM_i) \mid C_{4k}$, with $k \in 1..i * j$ then for $A \mid \Gamma_{1i} + \Gamma_{2j} \vdash_{\cap G} e_1\ e_2 :$

$T_{3i} \mid C_{1i} \cup C_{2j} \cup C_{3i} \cup C_{4k}$, such that $\exists S_k \, . \, S_k \models C_{1i} \cup C_{2j} \cup C_{3i} \cup C_{4k}$ then 2.a) for each $x \in dom(\Gamma) \cap dom(\Gamma_{1i} + \Gamma_{2j})$ we have that $\Gamma(x) \leq S_k(\Gamma_{1i} + \Gamma_{2j})(x)$, and 2.b) $S_1(T_{13}) \cap \ldots \cap S_{n' * m'}(T_{n'3}) \leq T$. We define $dom_{\triangleright}$ as $dom_{\triangleright}(Dyn) = Dyn$ and $dom_{\triangleright}(T_1 \to T_2) = T_1$ and $cod_{\triangleright}$ as $cod_{\triangleright}(Dyn) = Dyn$ and $cod_{\triangleright}(T_1 \to T_2) = T_2$. Since $cod_{\triangleright}(PM) = T$, we want to prove that $S_k(T_{i3}) \leq cod_{\triangleright}(S_{i1}(PM_i))$.

By Definition 1, rule 4, we have that $\Gamma(x) \leq (S_{1i}(\Gamma_{1i}) + S_{2j}(\Gamma_{2j}))(x)$. Since substitutions in S_{1i} don't affect Γ_{2j} and substitutions in S_{2j} don't affect Γ_{1i}, then $\Gamma(x) \leq (S_{1i} \circ S_{2j}(\Gamma_{1i} + \Gamma_{2j}))(x)$. For an $S_{3i} \models C_{3i}$ and $S_{4k} \models C_{4k}$, S_{3i} doesn't affect S_{2j}.

There are 3 possibilities:

* $PM_i = X$. Proof for 1. We have that exists $A \mid \Gamma_1 \vdash_{\cap G} e_1 : PM' \mid C_1$ such that $\exists S_1 \, . \, S_1 \models C_1$ and exists $A \mid \Gamma_2 \vdash_{\cap G} e_2 : T'' \mid C_2$ such that $\exists S_2 \, . \, S_2 \models C_2$, and for $cod(X) \doteq X_2 \mid \{X \doteq X_1 \to X_2\}$ and $T'' \dot{\lesssim} dom(PM') \mid \{X \doteq X_3 \to X_4, T'' \dot{\lesssim} X_3\}$ then, by rule C-App, $A \mid \Gamma_1 + \Gamma_2 \vdash_{\cap G} e_1 \; e_2 : T_3 \mid C_1 \cup C_2 \cup \{X \doteq X_1 \to X_2\} \cup \{X \doteq X_3 \to X_4, T'' \dot{\lesssim} X_3\}$. We now have to prove that $\exists S \, . \, S \models C_1 \cup C_2 \cup \{X \doteq X_1 \to X_2\} \cup \{X \doteq X_3 \to X_4, T'' \dot{\lesssim} X_3\}$. Since $S_2(T'') \leq T'_1 \cap \ldots \cap T'_n$, and $T'_1 \lesssim T_1$ and $\ldots$ and $T'_n \lesssim T_n$ and $T_1 \cap \ldots \cap T_n \leq dom_{\triangleright} S_1(PM')$, then $S_2(T'') \lesssim dom_{\triangleright}(S_1(PM'))$. Therefore, it is proved.

  Proof for 2. For all $i \in 1..n'$, $j \in 1..m'$, such that $A \mid \Gamma_{1i} \vdash_{\cap G} e_1 : PM_i \mid C_{1i}$ and $\exists S_{1i} \, . \, S_{1i} \models C_{1i}$, $A \mid \Gamma_{2j} \vdash_{\cap G} e_2 : T''_j \mid C_{2j}$ and $\exists S_{2j} \, . \, S_{2j} \models C_{2j}$, $cod(PM_i) \doteq T_{3i} \mid C_{3i}$ and $T''_j \dot{\lesssim} dom(PM_i) \mid C_{4k}$, then $A \mid \Gamma_{1i} + \Gamma_{2j} \vdash_{\cap G} e_1 \; e_2 : T_{3i} \mid C_{1i} \cup C_{2j} \cup C_{3i} \cup C_{4k}$, with $k \in 1..i * j$.

  Since PM_i is a type variable, then there exists a term variable x such that $PM_i = \Gamma_{1i}(x)$ and so we have that $C_{3i} = \{X \doteq X_1 \to X_2\}$ and $C_{k4} = \{X \doteq X_3 \to X_4, T''_j \dot{\lesssim} X_3\}$. As $\Gamma(x) \leq S_{1i}(X)$ and, since we are dealing with an expression application, $\Gamma(x) = T_1 \to T$ for some simple types T_1 and T, then $T_1 \to T \leq S_{1i}(X)$. Since substitutions don't introduce intersection types, then $T_1 \to T = S_{1i}(X)$.

  Proof for 2.a). If $S_k \models T''_j \dot{\lesssim} X_3$, then by Definition 3, $S_k(T''_j) \lesssim S_k(X_3)$. If $T''_j \in cod(S_{2j}(\Gamma_{2j}))$ and T''_j is static, then $S_{2j}(\Gamma_{2j})(x) \leq S_k(\Gamma_{2j})(x)$. Also, since $X \in cod(S_{i1}(\Gamma_{i1}))$, then $S_{i1}(\Gamma_{i1}) \leq S_k(\Gamma_{i1})$. For a S_k such that $S_k \models C_{i1} \cup C_{j2} \cup C_{i3} \cup C_{k4}$, $\Gamma(x) \leq S_k(\Gamma_{i1} + \Gamma_{j2})(x)$.

  Proof for 2.b). We have that $T = cod_{\triangleright}(S_{i1}(PM_i))$ and $S_k(T_{i3}) = T$.
* $PM_i = T_3 \to T_4$. We have that exists $A \mid \Gamma_1 \vdash_{\cap G} e_1 : PM' \mid C_1$ such that $\exists S_1 \, . \, S_1 \models C_1$ and exists $A \mid \Gamma_2 \vdash_{\cap G} e_2 : T'' \mid C_2$

such that $\exists S_2 \ . \ S_2 \models C_2$, and for $cod(T_3 \rightarrow T_4) \doteq T_4 \mid \{\}$ and $T'' \dot{\lesssim} dom(T_3 \rightarrow T_4) \mid \{T'' \dot{\lesssim} T_3\}$ then, by rule C-App, $A \mid \Gamma_1 + \Gamma_2 \vdash_{\cap G} e_1 \ e_2 : T_4 \mid C_1 \cup C_2 \cup \{T'' \dot{\lesssim} T_3\}$. We now have to prove that $\exists S \ . \ S \models C_1 \cup C_2 \cup \{T'' \dot{\lesssim} T_3\}$. Since $S_2(T'') \leq T'_1 \cap \ldots \cap T'_n$, and $T'_1 \lesssim T_1$ and $\ldots$ and $T'_n \lesssim T_n$ and $T_1 \cap \ldots \cap T_n \leq S_1(T_3)$, then $S_2(T'') \lesssim S_1(T_3)$. Therefore, it is proved.

For all $i \in 1..n'$, $j \in 1..m'$, such that $A \mid \Gamma_{1i} \vdash_{\cap G} e_1 : PM_i \mid C_{1i}$ and $\exists S_{1i} \ . \ S_{1i} \models C_{1i}$, $A \mid \Gamma_{2j} \vdash_{\cap G} e_2 : T''_j \mid C_{2j}$ and $\exists S_{2j} \ . \ S_{2j} \models C_{2j}$, $cod(PM_i) \doteq T_{3i} \mid C_{3i}$ and $T''_j \dot{\lesssim} dom(PM_i) \mid C_{4k}$, then $A \mid \Gamma_{1i} + \Gamma_{2j} \vdash_{\cap G} e_1 \ e_2 : T_{3i} \mid C_{1i} \cup C_{2j} \cup C_{3i} \cup C_{4k}$, with $k \in 1..i * j$.

Proof for 2.a). S_{i3} doesn't affect Γ_{i1} and Γ_{j2}. We don't allow variables in annotations in lambda abstractions. If $T_3 = Dyn$ or $T''_j = Dyn$ then $[] \models T''_j \dot{\lesssim} T_3$ and so, $\Gamma(x) \leq S_k(\Gamma_{i1} + \Gamma_{j2})(x)$. One way that $PM_i = T_3 \rightarrow T_4$ is if e_1 is a term variable and T_3 is a type variable, and so $T_3 \notin \Gamma_{i1}$ then $\Gamma(x) \leq S_k(\Gamma_{i1} + \Gamma_{j2})(x)$. Another way that $PM_i = T_3 \rightarrow T_4$ is if e_1 is a lambda abstraction and $T_3 \rightarrow T_4 \in \Gamma_{i1}$, and so T_3 is not a type variable, then $\Gamma(x) \leq S_k(\Gamma_{i1}+\Gamma_{j2})(x)$. Therefore, if $T''_j \in \Gamma_{j2}$, and as $S_k \models T''_j \dot{\lesssim} T_3$ then $\Gamma(x) \leq S_k(\Gamma_{i1}+\Gamma_{j2})(x)$.

Proof for 2.b). We have that $T_{i3} = T_4$, then $cod_{\triangleright}(S_{i1}(PM_i)) = S_{i1}(T_{i3})$. We want to prove that $S_i(T_{i3}) \leq S_{i1}(T_{i3})$. If T_{i3} is not a variable, then $S_i(T_{i3}) = S_{i1}(T_{i3})$. If T_{i3} is a variable, then either $T_{i3} \neq T_3$, in which case S_k doesn't affect $S_{i1}(T_4)$ and so $S_{i1}(T_4) = S_k(T_4)$. Otherwise, $T_3 = T_4 = T_{i3}$. Therefore, as $S_k \models T''_j \dot{\lesssim} T_4$. So, $S_k(T_4) \lesssim S_{i1}(T_4)$. Since S_k doesn't have a subtitution that turns T_4 into Dyn, then by Lemma 10, $S_k(T_4) \leq S_{i1}(T_4)$.

* $PM_i = Dyn$. Proof for 1. We have that exists $A \mid \Gamma_1 \vdash_{\cap G} e_1 : Dyn \mid C_1$ such that $\exists S_1 \ . \ S_1 \models C_1$ and exists $A \mid \Gamma_2 \vdash_{\cap G} e_2 : T'' \mid C_2$ such that $\exists S_2 \ . \ S_2 \models C_2$, and for $cod(Dyn) \doteq Dyn \mid \{\}$ and $T'' \dot{\lesssim} dom(Dyn) \mid \{T'' \dot{\lesssim} Dyn\}$ then, by rule C-App, $A \mid \Gamma_1 + \Gamma_2 \vdash_{\cap G} e_1 \ e_2 : Dyn \mid C_1 \cup C_2 \cup \{T'' \dot{\lesssim} Dyn\}$. Since $\exists S \ . \ S \models C_1 \cup C_2 \cup \{T'' \dot{\lesssim} Dyn\}$, it is proved.

  Proof for 2. For all $i \in 1..n'$, $j \in 1..m'$, such that $A \mid \Gamma_{1i} \vdash_{\cap G} e_1 : PM_i \mid C_{1i}$ and $\exists S_{1i} \ . \ S_{1i} \models C_{1i}$, $A \mid \Gamma_{2j} \vdash_{\cap G} e_2 : T''_j \mid C_{2j}$ and $\exists S_{2j} \ . \ S_{2j} \models C_{2j}$, $cod(PM_i) \doteq T_{3i} \mid C_{3i}$ and $T''_j \dot{\lesssim} dom(PM_i) \mid C_{4k}$, then $A \mid \Gamma_{1i} + \Gamma_{2j} \vdash_{\cap G} e_1 \ e_2 : T_{3i} \mid C_{1i} \cup C_{2j} \cup C_{3i} \cup C_{4k}$, with $k \in 1..i * j$.

  Proof for 2.a). For $A \mid \Gamma_{1i}+\Gamma_{2j} \vdash_{\cap G} e_1 \ e_2 : T_{3i} \mid C_{1i} \cup C_{2j} \cup C_{3i} \cup C_{4k}$, with $k \in 1..i * j$ such that $S_k \models C_{1i} \cup C_{2j} \cup C_{3i} \cup C_{4k}$, we have that $C_{i3} = \{\}$ and $C_{k4} = \{T''_j \dot{\lesssim} Dyn\}$. Therefore, $S_k = S_1 \circ S_2$ and then

$\Gamma(x) \leq S_k(\Gamma_{i1} + \Gamma_{j2})(x)$.

Proof for 2.b). We have that $cod_{\triangleright}(S_{i1}(PM_i)) = Dyn$ and $S_i(T_i3) = Dyn$.

- Using rule C-App$\cap$. By the induction hypothesis on 1., exists $A \mid \Gamma' \vdash_{\cap G} e_1 : T_1 \cap \ldots \cap T_m \to T_0 \mid C$ such that $\exists S \,.\, S \models C$ and exists $A \mid \Gamma'' \vdash_{\cap G} e_2 : T'' \mid C''$ such that $\exists S'' \,.\, S'' \models C''$ and $\ldots$ and exists $A \mid \Gamma'' \vdash_{\cap G} e_2 : T'' \mid C''$ such that $\exists S'' \,.\, S'' \models C''$.

  By the induction hypothesis on 2., we have that for $A \mid \Gamma_1 \vdash_{\cap G} e_1 : T_{11} \cap \ldots \cap T_{1m^1} \to T_{10} \mid C_1$ such that $\exists S_1 \,.\, S_1 \models C_1$ and $\ldots$ and for $A \mid \Gamma_{n'} \vdash_{\cap G} e_1 : T_{n'1} \cap \ldots \cap T_{n'm^{n'}} \to T_{n'0} \mid C_{n'}$ such that $\exists S_{n'} \,.\, S_{n'} \models C_{n'}$ then for each $x \in dom(\Gamma) \cap dom(\sum_{i=1}^{n'} \Gamma_i)$, we have that $\Gamma(x) \leq S_i(\Gamma_i(x))$ and $\bigcap_{i=1}^{n'} S_i(T_{i1} \cap \ldots \cap T_{im^i} \to T_{i0}) \leq PM$.

  Also, by the induction hypothesis on 2., we have that for $A \mid \Gamma'_1 \vdash_{\cap G} e_2 : T''_1 \mid C'_1$ such that $\exists S'_1 \,.\, S'_1 \models C'_1$ and $\ldots$ and for $A \mid \Gamma'_k \vdash_{\cap G} e_2 : T''_k \mid C'_k$ such that $\exists S'_k \,.\, S'_k \models C'_k$ then for each $x \in dom(\Gamma) \cap dom(\sum_{l=1}^{k} \Gamma'_i)$, we have that $\Gamma(x) \leq S'_l(\Gamma'_l(x))$ and $\bigcap_{l=1}^{k} S'_l(T''_l) \leq T'_1 \cap \ldots \cap T'_n$.

  Proof for 1. If $S(T_1 \cap \ldots \cap T_m \to T_0) \leq PM$, then by Definition 1 and $\triangleright$, $PM = T_1 \cap \ldots \cap T_n \to T$. Therefore, $T_1 \cap \ldots \cap T_n \leq S(T_1 \cap \ldots \cap T_m)$ and $S(T_0) \leq T$. We have that $S''(T'') \leq T'_1 \cap \ldots \cap T'_n$ and $T'_1 \lesssim T_1$ and $\ldots$ and $T'_n \lesssim T_n$ and $T_1 \cap \ldots \cap T_n \leq S(T_1 \cap \ldots \cap T_m)$. Therefore, we have that $S''(T'') \lesssim S(T_1)$ and $\ldots$ and $S''(T'') \lesssim S(T_m)$. Therefore, we have that $A \mid \Gamma' + \Gamma'' + \ldots + \Gamma'' \vdash_{\cap G} e_1\ e_2 : T_0 \mid C \cup C'' \cup \{T'' \dot{\lesssim} T_1\} \cup \ldots \cup C'' \cup \{T'' \dot{\lesssim} T_m\}$ such that $S \circ S'' \circ \ldots \circ S'' \models C \cup C'' \cup \{T'' \dot{\lesssim} T_1\} \cup \ldots \cup C'' \cup \{T'' \dot{\lesssim} T_m\}$.

  Proof for 2. For all $i \in 1..n'$, $j \in 1..m^i$, $l, l' \in 1..k$, such that $A \mid \Gamma_i \vdash_{\cap G} e_1 : T_{i1} \cap \ldots \cap T_{im^i} \to T_{i0} \mid C_i$ such that $\exists S_i \,.\, S_i \models C_i$, $A \mid \Gamma'_l \vdash_{\cap G} e_2 : T''_l \mid C'_l$ such that $\exists S'_l \,.\, S'_l \models C'_l$ and $\ldots$ and $A \mid \Gamma'_{l'} \vdash_{\cap G} e_2 : T''_{l'} \mid C'_{l'}$ such that $\exists S'_{l'} \,.\, S'_l \models C'_{l'}$ then $A \mid \Gamma_i + \Gamma'_l + \ldots + \Gamma'_{l'} \vdash_{\cap G} e_1\ e_2 : T_{i0} \mid C_i \cup C'_l \cup \{T''_l \dot{\lesssim} T_{i1}\} \cup \ldots \cup C'_{l'} \cup \{T''_{l'} \dot{\lesssim} T_{im^i}\}$.

  Proof for 2.a). By Definition 1, rule 4, we have that $\Gamma(x) \leq (S_i(\Gamma_i) + S'_l(\Gamma'_l) + \ldots + S'_{l'}(\Gamma'_{l'}))(x)$. Since substitutions in S_i and S'_l and $\ldots$ and $S'_{l'}$, don't affect each other, then $\Gamma(x) \leq S_i \circ S'_l \circ \ldots \circ S'_{l'}(\Gamma_i + \Gamma'_l + \ldots + \Gamma'_{l'})(x)$. For all $i \in 1..n'$, $j \in 1..m^i$, $l, l' \in 1..k$, for $A \mid \Gamma_i + \Gamma'_l + \ldots + \Gamma'_{l'} \vdash_{\cap G} e_1\ e_2 : T_{i0} \mid C_i \cup C'_l \cup \{T''_l \dot{\lesssim} T_{i1}\} \cup \ldots \cup C'_{l'} \cup \{T''_{l'} \dot{\lesssim} T_{im^i}\}$ such that $\exists S_i \circ S'_l \circ S''_l \circ \ldots \circ S'_{l'} \circ S''_{l'} \,.\, S_i \circ S'_l \circ S''_l \circ \ldots \circ S'_{l'} \circ S''_{l'} \models C_i \cup C'_l \cup \{T''_l \dot{\lesssim} T_{i1}\} \cup \ldots \cup C'_{l'} \cup \{T''_{l'} \dot{\lesssim} T_{im^i}\}$, with $S''_l \models T''_l \dot{\lesssim} T_{i1}$ and $\ldots$ and $S''_{l'} \models T''_{l'} \dot{\lesssim} T_{im^i}$, then we have several possibilities. If either $T''_l = Dyn$ or $T_{ij} = Dyn$, then $[] \models T''_l \dot{\lesssim} T_{ij}$, and therefore

$\Gamma(x) \leq S_i \circ S'_l \circ S''_l \circ \ldots \circ S'_{l'} \circ S''_{l'}(\Gamma_i + \Gamma'_l + \ldots + \Gamma'_{l'})(x)$. If $T''_l \in cod(\Gamma'_l)$, since $S''_l \models T''_l \dot{\lesssim} T_{ij}$, then $\Gamma(x) \leq S_i \circ S'_l \circ S''_l \circ \ldots \circ S'_{l'} \circ S''_{l'}(\Gamma_i + \Gamma'_l + \ldots + \Gamma'_{l'})(x)$. If e_1 is a lambda abstraction, then $T_{im^i} \notin cod(\Gamma_i)$. If e_1 is a term variable, then $T_{ij} \to T''' \in \Gamma_i$, for some T'''. Since $S''_l \models T''_l \dot{\lesssim} T_{ij}$, then $\Gamma(x) \leq S_i \circ S'_l \circ S''_l \circ \ldots \circ S'_{l'} \circ S''_{l'}(\Gamma_i + \Gamma'_l + \ldots + \Gamma'_{l'})(x)$.

Proof for 2.b). If $S_1(T_{11} \cap \ldots \cap T_{1m^1} \to T_{10}) \cap \ldots \cap S_{n'}(T'_{n'1} \cap \ldots \cap T_{n'm^{n'}} \to T_{n'0}) \leq PM$, then by Definition 1 and $\rhd$, $PM = T_1 \cap \ldots \cap T_n \to T$. Therefore, $S_1(T_{10}) \cap \ldots \cap S_{n'}(T_{n'0}) \leq T$. Since T_{i0} is not affected by substitutions besides S_i, then $\bigcap_{i=1}^{n'}(\bigcap_{l=1}^{k} \ldots \bigcap_{l'=1}^{k} S_i \circ S'_l \circ S''_l \circ \cdots \circ S'_{l'} \circ S''_{l'}(T_{i0})) \leq T$.

- Rule T-Gen. If $\Gamma \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$ then $\Gamma \vdash_{\cap G} e : T_1$ and $\ldots$ and $\Gamma \vdash_{\cap G} e : T_n$. By the induction hypothesis on 1., exists $A \mid \Gamma_1 \vdash_{\cap G} e : T'_1 \mid C_1$ such that $\exists S_1 \,.\, S_1 \models C_1$ and $\ldots$ and exists $A \mid \Gamma_n \vdash_{\cap G} e : T'_n \mid C_n$ such that $\exists S_n \,.\, S_n \models C_n$.

  By the induction hypothesis on 2., we have that for $A \mid \Gamma_{11} \vdash_{\cap G} e : T'_{11} \mid C_{11}$ such that $\exists S_{11} \,.\, S_{11} \models C_{11}$ and $\ldots$ and for $A \mid \Gamma_{1m^1} \vdash_{\cap G} e : T'_{1m^1} \mid C_{1m^1}$ such that $\exists S_{1m^1} \,.\, S_{1m^1} \models C_{1m^1}$ then for each $x \in dom(\Gamma) \cap dom(\sum_{j=1}^{m^1} \Gamma_{1j})$, we have that $\Gamma(x) \leq S_{1j}(\Gamma_{1j}(x))$, $\forall j \in 1..m^1$, and $S_{11}(T'_{11}) \cap \ldots \cap S_{1m^1}(T'_{1m^1}) \leq T_1$ and $\ldots$ and we have that for $A \mid \Gamma_{n1} \vdash_{\cap G} e : T'_{n1} \mid C_{n1}$ such that $\exists S_{n1} \,.\, S_{n1} \models C_{n1}$ and $\ldots$ and for $A \mid \Gamma_{nm^n} \vdash_{\cap G} e : T'_{nm^n} \mid C_{nm^n}$ such that $\exists S_{nm^n} \,.\, S_{nm^n} \models C_{nm^n}$ then for each $x \in dom(\Gamma) \cap dom(\sum_{j=1}^{m^n} \Gamma_{nj})$, we have that $\Gamma(x) \leq S_{nj}(\Gamma_{nj}(x))$, $\forall j \in 1..m^n$, and $S_{n1}(T'_{n1}) \cap \ldots \cap S_{nm^n}(T'_{nm^n}) \leq T_n$.

  Proof for 2.b). By Definition 1, we have that $S_{11}(T'_{11}) \cap \ldots \cap S_{1m^1}(T'_{1m^1}) \cap \ldots \cap S_{n1}(T'_{n1}) \cap \ldots \cap S_{nm^n}(T'_{nm^n}) \leq T_1 \cap \ldots \cap T_n$.
- Rule T-Inst. If $\Gamma_1 \vdash_{\cap G} e : T_i$ then $\Gamma_1 \vdash_{\cap G} e : T_1 \cap \ldots \cap T_n$. By the induction hypothesis on 1., exists $A \mid \Gamma_2 \vdash_{\cap G} e : T' \mid C$ such that $\exists S \,.\, S \models C$.

  By the induction hypothesis on 2., we have that for $A \mid \Gamma_{21} \vdash_{\cap G} e : T'_1 \mid C_1$ such that $\exists S_1 \,.\, S_1 \models C_1$ and $\ldots$ and for $A \mid \Gamma_{2n} \vdash_{\cap G} e : T'_n \mid C_n$ such that $\exists S_n \,.\, S_n \models C_n$ then for each $x \in dom(\Gamma_1) \cap dom(\sum_{i=1}^{n} \Gamma_{2i})$, we have $\Gamma_1(x) \leq S_i(\Gamma_{2i}(x))$, $\forall i \in 1..n$, and $S_1(T'_1) \cap \ldots \cap S_n(T'_n) \leq T_1 \cap \ldots \cap T_n$.

  Proof for 2.b). As, by definition 1, $T_1 \cap \ldots \cap T_n \leq T_i$, by transitivity, $S_1(T'_1) \cap \ldots \cap S_n(T'_n) \leq T_i$.

Lemma 3 (Unification Soundness). *If $C \Rightarrow S$ then $S \models C$.*

Proof. We proceed by induction on the length of the derivation tree of $C \Rightarrow S$.

Base cases:

- Rule Em. If $\emptyset \Rightarrow \emptyset$, then by definition 3, $\emptyset \models \emptyset$.

Induction step:

- Rule CS-DynL. If $\{Dyn \mathrel{\dot{\lesssim}} T\} \cup C \Rightarrow S$ then $C \Rightarrow S$. By the induction hypothesis, $S \models C$. Since $S(Dyn) \lesssim S(T)$ then $S \models Dyn \mathrel{\dot{\lesssim}} T$. Therefore, by definition 3, $S \models \{Dyn \mathrel{\dot{\lesssim}} T\} \cup C$.
- Rule CS-DynR. If $\{T \mathrel{\dot{\lesssim}} Dyn\} \cup C \Rightarrow S$ then $C \Rightarrow S$. By the induction hypothesis, $S \models C$. Since $S(T) \lesssim S(Dyn)$ then $S \models T \mathrel{\dot{\lesssim}} Dyn$. Therefore, by definition 3, $S \models \{T \mathrel{\dot{\lesssim}} Dyn\} \cup C$.
- Rule CS-Refl. If $\{T \mathrel{\dot{\lesssim}} T\} \cup C \Rightarrow S$ then $C \Rightarrow S$. By the induction hypothesis, $S \models C$. Since $S(T) \lesssim S(T)$, then $S \models T \mathrel{\dot{\lesssim}} T$. Therefore, by definition 3, $S \models \{T \mathrel{\dot{\lesssim}} T\} \cup C$.
- Rule CS-Inst. If $\{T_1 \cap \ldots \cap T_n \mathrel{\dot{\lesssim}} T_1 \cap \ldots \cap T_m\} \cup C \Rightarrow S$ then $C \Rightarrow S$. By the induction hypothesis, $S \models C$. Since $S(T_1 \cap \ldots \cap T_n) \lesssim S(T_1 \cap \ldots \cap T_m)$, then $S \models T_1 \cap \ldots \cap T_n \mathrel{\dot{\lesssim}} T_1 \cap \ldots \cap T_m$. Therefore, by definition 3, $S \models \{T_1 \cap \ldots \cap T_n \mathrel{\dot{\lesssim}} T_1 \cap \ldots \cap T_m\} \cup C$.
- Rule CS-Assoc. If $\{(T \to T_1) \cap \ldots \cap (T \to T_n) \mathrel{\dot{\lesssim}} T \to T_1 \cap \ldots \cap T_n\} \cup C \Rightarrow S$ then $C \Rightarrow S$. By the induction hypothesis, $S \models C$. Since $S((T \to T_1) \cap \ldots \cap (T \to T_n)) \lesssim S(T \to T_1 \cap \ldots \cap T_n)$, then $S \models (T \to T_1) \cap \ldots \cap (T \to T_n) \mathrel{\dot{\lesssim}} T \to T_1 \cap \ldots \cap T_n$. Therefore, by definition 3, $S \models \{(T \to T_1) \cap \ldots \cap (T \to T_n) \mathrel{\dot{\lesssim}} T \to T_1 \cap \ldots \cap T_n\} \cup C$.
- Rule CS-Arrow. If $\{T_1 \to T_2 \mathrel{\dot{\lesssim}} T_3 \to T_4\} \cup C \Rightarrow S$ then $\{T_3 \mathrel{\dot{\lesssim}} T_1, T_2 \mathrel{\dot{\lesssim}} T_4\} \cup C \Rightarrow S$. By the induction hypothesis, $S \models \{T_3 \mathrel{\dot{\lesssim}} T_1, T_2 \mathrel{\dot{\lesssim}} T_4\} \cup C$. Since $S \models \{T_3 \mathrel{\dot{\lesssim}} T_1, T_2 \mathrel{\dot{\lesssim}} T_4\}$, then $S(T_3) \lesssim S(T_1)$ and $S(T_2) \lesssim S(T_4)$. Therefore, by definition 2, $S(T_1) \to S(T_2) \lesssim S(T_3) \to S(T_4)$. Therefore, $S(T_1 \to T_2) \lesssim S(T_3 \to T_4)$. By definition 3, $S \models \{T_1 \to T_2 \mathrel{\dot{\lesssim}} T_3 \to T_4\}$. Therefore, by definition 3, $S \models \{T_1 \to T_2 \mathrel{\dot{\lesssim}} T_3 \to T_4\} \cup C$.
- Rule CS-InstR. If $\{T \mathrel{\dot{\lesssim}} T_1 \cap \ldots \cap T_n\} \cup C \Rightarrow S$ then $\{T \mathrel{\dot{\lesssim}} T_1 \wedge \ldots \wedge T \mathrel{\dot{\lesssim}} T_n\} \cup C \Rightarrow S$. By the induction hypothesis, $S \models \{T \mathrel{\dot{\lesssim}} T_1, \ldots, T \mathrel{\dot{\lesssim}} T_n\} \cup C$. Since $S \models \{T \mathrel{\dot{\lesssim}} T_1, \ldots, T \mathrel{\dot{\lesssim}} T_n\}$, then by definition 3, $S(T) \lesssim S(T_1) \wedge \ldots \wedge S(T) \lesssim S(T_n)$. Therefore, by definition 2, $S(T) \lesssim S(T_1) \cap \ldots \cap S(T_n)$. Therefore, $S(T) \lesssim S(T_1 \cap \ldots \cap T_n)$. By definition 3, $S \models T \mathrel{\dot{\lesssim}} T_1 \cap \ldots \cap T_n$. Therefore, $S \models \{T \mathrel{\dot{\lesssim}} T_1 \cap \ldots \cap T_n\} \cup C$.
- Rule CS-ArrowL. If $\{T_1 \to T_2 \mathrel{\dot{\lesssim}} T\} \cup C \Rightarrow S$ then $\{T_3 \mathrel{\dot{\lesssim}} T_1, T_2 \mathrel{\dot{\lesssim}} T_4, T = T_3 \to T_4\} \cup C \Rightarrow S$. By the induction hypothesis, $S \models \{T_3 \mathrel{\dot{\lesssim}} T_1, T_2 \mathrel{\dot{\lesssim}} T_4, T \doteq T_3 \to T_4\} \cup C$. Since $S \models \{T_3 \mathrel{\dot{\lesssim}} T_1, T_2 \mathrel{\dot{\lesssim}} T_4, T \doteq T_3 \to T_4\}$, then by definition 3, $S(T_3) \lesssim S(T_1)$ and $S(T_2) \lesssim S(T_4)$ and $S(T) = S(T_3 \to T_4)$. By definition of $\lesssim$, $S(T_1) \to S(T_2) \lesssim S(T_3) \to S(T_4)$. Therefore, $S(T_1 \to T_2) \lesssim S(T_3 \to T_4)$. Since $S(T) = S(T_3 \to T_4)$, then $S(T_1 \to T_2) \lesssim S(T)$. Therefore, by definition 3, $S \models T_1 \to T_2 \mathrel{\dot{\lesssim}} T$. Therefore, $S \models \{T_1 \to T_2 \mathrel{\dot{\lesssim}} T\} \cup C$.
- Rule CS-ArrowR. If $\{T \mathrel{\dot{\lesssim}} T_1 \to T_2\} \cup C \Rightarrow S$ then $\{T_1 \mathrel{\dot{\lesssim}} T_3, T_4 \mathrel{\dot{\lesssim}} T_2, T = T_3 \to T_4\} \cup C \Rightarrow S$. By the induction hypothesis, $S \models \{T_1 \mathrel{\dot{\lesssim}} T_3, T_4 \mathrel{\dot{\lesssim}} T_2, T \doteq T_3 \to T_4\} \cup C$. Since $S \models \{T_1 \mathrel{\dot{\lesssim}} T_3, T_4 \mathrel{\dot{\lesssim}} T_2, T \doteq T_3 \to T_4\}$, then by definition 3, $S(T_1) \lesssim S(T_3)$ and $S(T_4) \lesssim S(T_2)$ and $S(T) = S(T_3 \to T_4)$. By definition of $\lesssim$, $S(T_3) \to S(T_4) \lesssim S(T_1) \to S(T_2)$. Therefore, $S(T_3 \to T_4) \lesssim S(T_1 \to$

T_2). Since $S(T) = S(T_3 \to T_4)$, then $S(T) \lesssim S(T_1 \to T_2)$. Therefore, by definition 3, $S \models T \dot{\lesssim} T_1 \to T_2$. Therefore, $S \models \{T \dot{\lesssim} T_1 \to T_2\} \cup C$.
- Rule CS-Eq. If $\{T_1 \dot{\lesssim} T_2\} \cup C \Rightarrow S$ then $\{T_1 \doteq T_2\} \cup C \Rightarrow S$. By the induction hypothesis, $S \models \{T_1 \doteq T_2\} \cup C$. By definition 3, $S(T_1) = S(T_2)$. By definition 2, $S(T_1) \lesssim S(T_2)$. By definition 3, $S \models T_1 \dot{\lesssim} T_2$. Therefore, $S \models \{T_1 \dot{\lesssim} T_2\} \cup C$.
- Rule Eq-Refl. If $\{T \doteq T\} \cup C \Rightarrow S$ then $C \Rightarrow S$. By the induction hypothesis, $S \models C$. Since $S(T) = S(T)$, then by definition 3, $S \models T \doteq T$. Therefore, $S \models \{T \doteq T\} \cup C$.
- Rule Eq-Arrow. If $\{T_1 \to T_2 \doteq T_3 \to T_4\} \cup C \Rightarrow S$ then $\{T_1 \doteq T_3, T_2 \doteq T_4\} \cup C \Rightarrow S$. By the induction hypothesis, $S \models \{T_1 \doteq T_3, T_2 \doteq T_4\} \cup C$. By definition 3, $S(T_1) = S(T_3)$ and $S(T_2) = S(T_4)$. Then $S(T_1) \to S(T_2) = S(T_3) \to S(T_4)$. Therefore, $S(T_1 \to T_2) = S(T_3 \to T_4)$. By definition 3, $S \models T_1 \to T_2 \doteq T_3 \to T_4$. Therefore, $S \models \{T_1 \to T_2 \doteq T_3 \to T_4\} \cup C$.
- Rule Eq-VarR. If $\{T \doteq X\} \cup C \Rightarrow S$ then $\{X \doteq T\} \wedge C \Rightarrow S$. By the induction hypothesis, $S \models \{X \doteq T\} \cup C$. By definition 3, $S(X) = S(T)$. Then, $S(T) = S(X)$. By definition 3, $S \models T \doteq X$. Therefore, $S \models \{T \doteq X\} \cup C$.
- Rule Eq-VarL. If $\{X \doteq T\} \cup C \Rightarrow S \circ [X \mapsto T]$ then $[X \mapsto T]C \Rightarrow S$. By the induction hypothesis, $S \models [X \mapsto T]C$. Then, for each constraint of the form $T_1' \doteq T_2'$ or $T_1' \dot{\lesssim} T_2'$ in C, $S([X \mapsto T]T_1') = S([X \mapsto T]T_2')$ or $S([X \mapsto T]T_1') \leq S([X \mapsto T]T_2')$. Therefore, $S \circ [X \mapsto T](T_1') = S \circ [X \mapsto T](T_2')$ or $S \circ [X \mapsto T](T_1') \leq S \circ [X \mapsto T](T_2')$. Therefore, $S \circ [X \mapsto T] \models C$. It follows that $S \circ [X \mapsto T] \models \{X \doteq T\} \cup C$, because $S \circ [X \mapsto T](X) = S \circ [X \mapsto T](T)$. Therefore, $S \circ [X \mapsto T] \models \{X \doteq T\} \cup C$.

Lemma 4 (Unification Completeness). *If $S_1 \models C$ then $C \Rightarrow S_2$ for some S_2, and furthermore $S_1 = S \circ S_2$ for some S.*

Proof. We proceed by induction on the breakdown of constraint sets by the unification rules.

Base cases:

- Rule Em. If $S_1 \models \emptyset$ then $\emptyset \Rightarrow \emptyset$. As $S_1 = S \circ \emptyset$ for some S_1, it is proved.

Induction step:

- Rule CS-DynL. If $S_1 \models \{Dyn \dot{\lesssim} T\} \cup C$ then by definition 3, $S_1 \models C$. By the induction hypothesis, $C \Rightarrow S_2$ and $S_1 = S \circ S_2$. As $C \Rightarrow S_2$, then $\{Dyn \dot{\lesssim} T\} \cup C \Rightarrow S_2$.
- Rule CS-DynR. If $S_1 \models \{T \dot{\lesssim} Dyn\} \cup C$ then by definition 3, $S_1 \models C$. By the induction hypothesis, $C \Rightarrow S_2$ and $S_1 = S \circ S_2$. As $C \Rightarrow S_2$, then $\{T \dot{\lesssim} Dyn\} \cup C \Rightarrow S_2$.
- Rule CS-Refl. If $S_1 \models \{T \dot{\lesssim} T\} \cup C$ then by definition 3, $S_1 \models C$. By the induction hypothesis, $C \Rightarrow S_2$ and $S_1 = S \circ S_2$. As $C \Rightarrow S_2$, then $\{T \dot{\lesssim} T\} \cup C \Rightarrow S_2$.

- Rule CS-Inst. If $S_1 \models \{T_1 \cap \ldots \cap T_n \dot{\lesssim} T_1 \cap \ldots \cap T_m\} \cup C$ then by definition 3, $S_1 \models C$. By the induction hypothesis, $C \Rightarrow S_2$ and $S_1 = S \circ S_2$. As $C \Rightarrow S_2$, then $\{T_1 \cap \ldots \cap T_n \dot{\lesssim} T_1 \cap \ldots \cap T_m\} \cup C \Rightarrow S_2$.
- Rule CS-Assoc. If $S_1 \models \{(T \to T_1) \cap \ldots \cap (T \to T_n) \dot{\lesssim} T \to T_1 \cap \ldots \cap T_n\} \cup C$ then by definition 3, $S_1 \models C$. By the induction hypothesis, $C \Rightarrow S_2$ and $S_1 = S \circ S_2$. As $C \Rightarrow S_2$, then $\{(T \to T_1) \cap \ldots \cap (T \to T_n) \dot{\lesssim} T \to T_1 \cap \ldots \cap T_n\} \cup C \Rightarrow S_2$.
- Rule CS-Arrow. If $S_1 \models \{T_1 \to T_2 \dot{\lesssim} T_3 \to T_4\} \cup C$ then by definition 3, $S_1(T_1 \to T_2) \lesssim S_1(T_3 \to T_4)$ and $S_1 \models C$. Then, $S_1(T_1) \to S_1(T_2) \lesssim S_1(T_3) \to S_1(T_4)$ and by definition 2, $S_1(T_3) \lesssim S_1(T_1)$ and $S_1(T_2) \lesssim S_1(T_4)$. Then, by definition 3, $S_1 \models \{T_3 \dot{\lesssim} T_1, T_2 \dot{\lesssim} T_4\} \cup C$. By the induction hypothesis, $\{T_3 \dot{\lesssim} T_1, T_2 \dot{\lesssim} T_4\} \cup C \Rightarrow S_2$ and $S_1 = S \circ S_2$. Therefore, $\{T_1 \to T_2 \dot{\lesssim} T_3 \to T_4\} \cup C \Rightarrow S_2$.
- Rule CS-InstR. If $S_1 \models \{T \dot{\lesssim} T_1 \cap \ldots \cap T_n\} \cup C$ then by definition 3, $S_1(T) \lesssim S_1(T_1 \cap \ldots \cap T_n)$ and $S_1 \models C$. Therefore, by definition 2, $S_1(T) \lesssim S_1(T_1) \cap \ldots \cap S_1(T_n)$, and therefore, $S_1(T) \lesssim S_1(T_1)$ and $\ldots$ and $S_1(T) \lesssim S_1(T_n)$. By definition 3, $S_1 \models \{T \dot{\lesssim} T_1, \ldots, T \dot{\lesssim} T_n\} \cup C$. By the induction hypothesis, $\{T \dot{\lesssim} T_1, \ldots, T \dot{\lesssim} T_n\} \cup C \Rightarrow S_2$ and $S_1 = S \circ S_2$. Therefore, $\{T \dot{\lesssim} T_1 \cap \ldots \cap T_n\} \cup C \Rightarrow S_2$.
- Rule CS-ArrowL. If $S_1 \models \{T_1 \to T_2 \dot{\lesssim} T\} \cup C$ then, by definition 3, $S_1(T_1 \to T_2) \lesssim S_1(T)$ and $S_1 \models C$. Then, it exists a T_3 and T_4, such that $S_1(T) = S_1(T_3 \to T_4)$, so that $S_1(T_1 \to T_2) \lesssim S_1(T_3 \to T_4)$. By definition 2, $S_1(T_3) \lesssim S_1(T_1)$ and $S_1(T_2) \lesssim S_1(T_4)$. By definition 3, $S_1 \models T_3 \dot{\lesssim} T_1, T_2 \dot{\lesssim} T_4, T \doteq T_3 \to T_4 \cup C$. By the induction hypothesis, $\{T_3 \dot{\lesssim} T_1, T_2 \dot{\lesssim} T_4, T \doteq T_3 \to T_4\} \cup C \Rightarrow S_2$ and $S_1 = S \circ S_2$. Therefore, $\{T_1 \to T_2 \dot{\lesssim} T\} \cup C \Rightarrow S_2$.
- Rule CS-ArrowR. If $S_1 \models \{T \dot{\lesssim} T_1 \to T_2\} \cup C$ then, by definition 3, $S_1(T) \lesssim S_1(T_1 \to T_2)$ and $S_1 \models C$. Then, it exists a T_3 and T_4, such that $S_1(T) = S_1(T_3 \to T_4)$, so that $S_1(T_1 \to T_2) \lesssim S_1(T_3 \to T_4)$. By definition 2, $S_1(T_3) \lesssim S_1(T_1)$ and $S_1(T_2) \lesssim S_1(T_4)$. By definition 3, $S_1 \models T_3 \dot{\lesssim} T_1, T_2 \dot{\lesssim} T_4, T \doteq T_3 \to T_4 \cup C$. By the induction hypothesis, $\{T_3 \dot{\lesssim} T_1, T_2 \dot{\lesssim} T_4, T \doteq T_3 \to T_4\} \cup C \Rightarrow S_2$ and $S_1 = S \circ S_2$. Therefore, $\{T_1 \to T_2 \dot{\lesssim} T\} \cup C \Rightarrow S_2$.
- Rule CS-Eq. If $S_1 \models \{T_1 \dot{\lesssim} T_2\} \cup C$ and $T_1, T_2 \in \{Int, Bool\} \cup TVar$ then, by definition 3, $S_1(T_1) \lesssim S_1(T_2)$ and $S_1 \models C$. Therefore, by definition 2, $S_1(T_1) = S_1(T_2)$. Then, $S_1 \models \{T_1 \doteq T_2\}$. By the induction hypothesis, $\{T_1 \doteq T_2\} \Rightarrow S_2$ and $S_1 = S \circ S_2$. Therefore, $\{T_1 \dot{\lesssim} T_2\} \Rightarrow S_2$.
- Rule Eq-Refl. If $S_1 \models \{T \doteq T\} \cup C_1$ then, by definition 3, $S_1 \models C$. By the induction hypothesis, $C \Rightarrow S_2$ and $S_1 = S \circ S_2$. Therefore, $\{T \doteq T\} \cup C \Rightarrow S_2$.
- Rule Eq-Arrow. If $S_1 \models \{T_1 \to T_2 \doteq T_3 \to T_4\} \cup C$ then, by definition 3, $S_1(T_1 \to T_2) = S_1(T_3 \to T_4)$ and $S_1 \models C$. Then, $S_1(T_1) \to S_1(T_2) = S_1(T_3) \to S_1(T_4)$ and $S_1(T_1) = S_1(T_3)$ and $S_1(T_2) = S_1(T_4)$. Then, by definition 3, $S_1 \models \{T_1 \doteq T_3, T_2 \doteq T_4\} \cup C$. By the induction hypothesis,

$\{T_1 \doteq T_3, T_2 \doteq T_4\} \cup C \Rightarrow S_2$ and $S_1 = S \circ S_2$. Therefore, $\{T_1 \to T_2 \doteq T_3 \to T_4\} \cup C \Rightarrow S_2$.
- Rule Eq-VarR. If $S_1 \models \{T \doteq X\} \cup C$ then, by definition 3, $S_1(T) = S_1(X)$ and $S_1 \models C$. Then, $S_1(X) = S_1(T)$ and therefore, $S_1 \models \{X \doteq T\} \cup C$. By the induction hypothesis, $\{X \doteq T\} \cup C \Rightarrow S_2$ and $S_1 = S \circ S_2$. Therefore, $\{T \doteq X\} \cup C \Rightarrow S_2$.
- Rule Eq-VarL. If $S_1 \models \{X \doteq T\} \cup C$ then, by definition 3, $S_1(X) = S_1(T)$ and $S_1 \models C$. Then, $S_1 \models [X \mapsto T]C$. By the induction hypothesis, $[X \mapsto T]C \Rightarrow S_2$ and $S_1 = S \circ S_2$. Therefore, $\{X \doteq T\} \cup C \Rightarrow S_2 \circ [X \mapsto T]$ and $S_1 = S \circ S_2 \circ [X \mapsto T]$.

Lemma 5 (Unification Soundness). *If $G \mid C \Rightarrow S$ then $S \models C$.*

Proof. Only proofs for cases Em, CS-DynL, CS-DynR and Eq-VarL are included since proofs for other cases are straightforward adaptations from the proofs of Lemma 3. We proceed by induction on the length of the derivation tree of $G \mid C \Rightarrow S$.

Base cases:

- Rule Em. If $G \mid \emptyset \Rightarrow \overline{[Vars(G) \mapsto Dyn]}$, then by definition 3, $\overline{[Vars(G) \mapsto Dyn]} \models \emptyset$.

Induction step:

- Rule CS-DynL. If $G \mid \{Dyn \dot{\lesssim} T\} \cup C \Rightarrow S$ then $G \cup \{T\} \mid C \Rightarrow S$. By the induction hypothesis, $S \models C$. Since $S(Dyn) \lesssim S(T)$ then $S \models Dyn \dot{\lesssim} T$. Therefore, by definition 3, $S \models \{Dyn \dot{\lesssim} T\} \cup C$.
- Rule CS-DynR. If $G \mid \{T \dot{\lesssim} Dyn\} \cup C \Rightarrow S$ then $G \cup \{T\} \mid C \Rightarrow S$. By the induction hypothesis, $S \models C$. Since $S(T) \lesssim S(Dyn)$ then $S \models T \dot{\lesssim} Dyn$. Therefore, by definition 3, $S \models \{T \dot{\lesssim} Dyn\} \cup C$.
- Rule Eq-VarL. If $G \mid \{X \doteq T\} \cup C \Rightarrow S \circ [X \mapsto T]$ then $[X \mapsto T]G \mid [X \mapsto T]C \Rightarrow S$. By the induction hypothesis, $S \models [X \mapsto T]C$. Then, for each constraint of the form $T_1' \doteq T_2'$ or $T_1' \dot{\lesssim} T_2'$ in C, $S([X \mapsto T]T_1') = S([X \mapsto T]T_2')$ or $S([X \mapsto T]T_1') \leq S([X \mapsto T]T_2')$. Therefore, $S \circ [X \mapsto T](T_1') = S \circ [X \mapsto T](T_2')$ or $S \circ [X \mapsto T](T_1') \leq S \circ [X \mapsto T](T_2')$. Therefore, $S \circ [X \mapsto T] \models C$. It follows that $S \circ [X \mapsto T] \models \{X \doteq T\} \cup C$, because $S \circ [X \mapsto T](X) = S \circ [X \mapsto T](T)$. Therefore, $S \circ [X \mapsto T] \models \{X \doteq T\} \cup C$.

Lemma 6 (Unification Completeness). *If $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} \models C$ then $G \mid C \Rightarrow S_2$ for some S_2, and furthermore $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} = S \circ S_2$ for some S.*

Proof. Only proofs for cases Em, CS-DynL, CS-DynR and Eq-VarL are included since proofs for other cases are straightforward adaptations from the proofs of Lemma 4. We proceed by induction on the breakdown of constraint sets by the unification rules.

Base cases:

– Rule EM. If $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} \models \emptyset$ then $G \mid \emptyset \Rightarrow \overline{[Vars(G) \mapsto Dyn]}$. As $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} = S \circ \overline{[Vars(G) \mapsto Dyn]}$ for some S, it is proved.

Induction step:

– Rule CS-DYNL. If $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} \models \{Dyn \dot{\lesssim} T\} \cup C$ then by definition 3, $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} \models C$. By the induction hypothesis, $G \cup \{T\} \mid C \Rightarrow S_2$ and $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} = S \circ S_2$. As $G \cup \{T\} \mid C \Rightarrow S_2$, then $G \mid \{Dyn \dot{\lesssim} T\} \cup C \Rightarrow S_2$.
– Rule CS-DYNR. If $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} \models \{T \dot{\lesssim} Dyn\} \cup C$ then by definition 3, $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} \models C$. By the induction hypothesis, $G \cup \{T\} \mid C \Rightarrow S_2$ and $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} = S \circ S_2$. As $G \cup \{T\} \mid C \Rightarrow S_2$, then $G \mid \{T \dot{\lesssim} Dyn\} \cup C \Rightarrow S_2$.
– Rule EQ-VARL. If $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} \models \{X \doteq T\} \cup C$ then, by definition 3, $S_1 \circ \overline{[Vars(G) \mapsto Dyn]}(X) = S_1 \circ \overline{[Vars(G) \mapsto Dyn]}(T)$ and $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} \models C$. Then, $S_1 \models [X \mapsto T]C$. By the induction hypothesis, $[X \mapsto T]G \mid [X \mapsto T]C \Rightarrow S_2$ and $S_1 \circ \overline{[Vars(G) \mapsto Dyn]} = S \circ S_2$. Therefore, $G \mid \{X \doteq T\} \cup C \Rightarrow S_2 \circ [X \mapsto T]$.

Theorem 2 (Soundness). *If $(\Gamma, T, S) \in I(e)$ then $S(\Gamma) \vdash_{\cap G} S(e) : S(T)$.*

Proof. If $(\Gamma, T, S) \in I(e)$ then by Definition 5, $\emptyset \mid \Gamma \vdash_{\cap G} e : T \mid C$, $\emptyset \mid C \Rightarrow S$. By Lemma 5, $S \models C$. Therefore, by Lemma 1, $S(\Gamma) \vdash_{\cap G} S(e) : S(T)$.

Theorem 3 (Principal Typings). *If $\Gamma_1 \vdash_{\cap G} e : T_1$ then there are $\Gamma_{21}, \ldots, \Gamma_{2n}$, $T_{21}, \ldots, T_{2n}$, $S_{21}, \ldots, S_{2n}$ and $S_1, \ldots, S_n$ such that $((\Gamma_{21}, T_{21}, S_{21}), \ldots, (\Gamma_{2n}, T_{2n}, S_{2n})) = I(e)$ and, for each $x \in dom(\Gamma_1) \cap dom(\Gamma_{21} + \ldots + \Gamma_{2n})$, we have $\Gamma_1(x) \leq S_1 \circ S_{21}(\Gamma_{21}(x))$ and $\ldots$ and $\Gamma_1(x) \leq S_n \circ S_{2n}(\Gamma_{2n}(x))$ and $S_1 \circ S_{21}(T_{21}) \cap \ldots \cap S_n \circ S_{2n}(T_{2n}) \leq T_1$.*

Proof. If $\Gamma_1 \vdash_{\cap G} e : T_1$ then by Lemma 2, for $A \mid \Gamma_{21} \vdash_{\cap G} e : T_{21} \mid C_1$ such that $\exists S_{11} \,.\, S_{11} \models C_1$ and $\ldots$ and for $A \mid \Gamma_{2n} \vdash_{\cap G} e : T_{2n} \mid C_n$ such that $\exists S_{1n} \,.\, S_{1n} \models C_n$ then for each $x \in dom(\Gamma_1) \cap dom(\Gamma_{21} + \ldots + \Gamma_{2n})$, we have $\Gamma_1(x) \leq S_{11}(\Gamma_{21}(y))$ and $\ldots$ and $\Gamma_1(x) \leq S_{1n}(\Gamma_{2n}(y))$ and $S_{11}(T_{21}) \cap \ldots \cap S_{1n}(T_{2n}) \leq T_1$. By Lemma 6, $G_1 \mid C_1 \Rightarrow S_{21}$ for some S_{21} and furthermore $S_{11} = S_1 \circ S_{21}$, for some S_1 and $\ldots$ and $G_n \mid C_n \Rightarrow S_{2n}$ for some S_{2n} and furthermore $S_{1n} = S_n \circ S_{2n}$, for some S_n. As $A \mid \Gamma_{21} \vdash_{\cap G} e : T_{21} \mid C_1$ and $G_1 \mid C_1 \Rightarrow S_{21}$ and $\ldots$ and $A \mid \Gamma_{2n} \vdash_{\cap G} e : T_{2n} \mid C_n$ and $G_n \mid C_n \Rightarrow S_{2n}$, then by definition 5, $((\Gamma_{21}, T_{21}, S_{21}), \ldots, (\Gamma_{2n}, T_{2n}, S_{2n})) = I(e)$ and for each $x \in dom(\Gamma_1) \cap dom(\Gamma_{21} + \ldots + \Gamma_{2n})$, $\Gamma_1(x) \leq S_1 \circ S_{21}(\Gamma_{21}(x))$ and $\ldots$ and $\Gamma_1(x) \leq S_n \circ S_{2n}(\Gamma_{2n}(x))$ and $S_1 \circ S_{21}(T_{21}) \cap \ldots \cap S_n \circ S_{2n}(T_{2n}) \leq T_1$.

Lemma 8 (Termination of Constraint Solving). *$C \Rightarrow S$ terminates for every set of constraints C.*

Proof. A unification problem $C \Rightarrow S$ is solved if $C = \emptyset$. We define the following metrics with respect to the unification problem $C \Rightarrow S$:

- NICS is the number of unique intersection types in the left of an $\dot{\lesssim}$ constraint + the number of unique intersection types in the right of an $\dot{\lesssim}$ constraint
- NCCS is the number of type constructors in $\dot{\lesssim}$ constraints
- NCS is the number of $\dot{\lesssim}$ constraints
- NVEq is the number of different type variables in $\doteq$ constraints
- NCEq is the number of type constructors in $\doteq$ constraints
- NTXEq is the number of $\doteq$ constraints of the form $T \doteq X$
- NEq is the number of $\doteq$ constraints

We will prove termination by showing that both NCS and NEq reduce to 0.

The first part of the proof consists of reducing only $\dot{\lesssim}$ constraints. Termination of $C \Rightarrow S$, is proved by a measure function that maps the constraint set C to a tuple (NICS, NCCS, NCS). The following table shows that each step decreases the tuple w.r.t. the lexicographic order:

	NICS	NCCS	NCS
CS-DynL	$\geq$	$\geq$	$>$
CS-DynR	$\geq$	$\geq$	$>$
CS-Refl	$=$	$=$	$>$
CS-Inst	$>$		
CS-Assoc	$>$		
CS-Arrow	$=$	$>$	
CS-InstR	$>$		
CS-ArrowL	$\geq$	$>$	
CS-ArrowR	$\geq$	$>$	
CS-Eq	$=$	$=$	$>$

Note that the number of $\dot{\lesssim}$ constraints decreases to 0, leaving only $\doteq$ constraints in C.

The second part of the proof consists of reducing the remaining $\doteq$ constraints. Termination of $C \Rightarrow S$, where now only $\doteq$ are in C, is proved by a measure function that maps the constraint set C to a tuple (NVEq, NCEq, NTXEq, NEq). The following table shows that each step decreases the tuple w.r.t. the lexicographic order:

	NVEq	NCEq	NTXEq	NEq
Eq-Refl	$\geq$	$\geq$	$\geq$	$>$
Eq-Arrow	$=$	$>$		
Eq-VarR	$=$	$=$	$>$	
Eq-VarL	$>$			

Note that the number of $\doteq$ constraints decreases to 0, leaving C empty.