

INSTITUT FÜR INFORMATIK
der Ludwig Maximilian Universität München



Diplomarbeit

Algorithmic Subtyping for Higher-Order
Bounded Quantification Revisited

Dulma Rodriguez

Aufgabensteller: Prof. Martin Hofmann
Betreuer: Dr. Andreas Abel
Abgabetermin: 20. Dezember 2007

Erklärung

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 20. Dezember 2007

.....
(Unterschrift des Kandidaten)

Acknowledgement

I thank Andreas Abel for giving me the possibility of developing such a beautiful theory, and for all his support. I also thank Ralph Matthes for motivating me for the type theory. And I thank my parents for their love and support.

Contents

Introduction	vii
I. Higher Order Subtyping	1
1. System $F_{<}^\omega$	3
1.1. Constructors and kinds	4
1.2. Kinding and wellformed contexts	5
1.3. Similarity of contexts	7
1.4. Equality	8
1.5. Subtyping	12
1.6. Examples	15
2. Normalization of constructors	19
2.1. Equality of contexts	19
2.2. Hereditary substitution	21
2.3. Long normal forms	27
2.4. Characterization of long normal forms	29
2.5. Completeness of normalization	36
2.6. Summary	38
3. Algorithmic Subtyping	39
3.1. Soundness of algorithmic subtyping	41
3.2. Completeness of algorithmic subtyping	42
3.3. Termination of algorithmic subtyping	48
3.4. Example	51
II. Polarized Higher Order Subtyping	53
4. Polarized system $F_{<}^\omega$	55
4.1. Polarities	55
4.2. Kinds	57
4.3. Kinding	57
4.4. Equality	60
4.5. Subtyping	61

5. Algorithmic Polarized Subtyping	63
5.1. Weak head evaluation	63
5.2. Algorithm	63
5.3. Implementation in Haskell	65
Conclusions	67
Appendix	69

Introduction

A type may be viewed as a set of clothes (or suit of armor) that protects an underlying untyped representation from arbitrary or unintended use...
Cardelli, 1985.

This thesis is about types. It studies polymorphic type systems for modelling aspects of object-oriented languages. Thus we start with some motivation for the use of types in programming languages, and specially in object-oriented programming languages.

The first type systems in computer science, beginning in the 1950s in languages such as Fortran [Bac81], were introduced to improve the efficiency of numerical calculations by distinguishing between integer-valued arithmetic expressions and real-valued ones. In general, the most obvious benefit of type systems is that they allow early detection of some programming errors. Types are also useful when reading programs. The type declarations in procedure headers and module interfaces constitute a form of documentation. In automated theorem proving, type systems — usually very powerful ones — are used to represent logical propositions and proofs. Several proofs assistants, including Lego [Pol94] and Coq [BBC⁺97] are directly based on type theory.

For all these reasons the design of expressive, yet safe and decidable type systems is a major theme in the evolution of programming languages and an important contribution of theoretical computer science. The development of safe, expressive type systems is especially challenging for object-oriented languages. In the following we describe some of their features which have to be modelled in the type system.

Classes. Object-oriented languages might be defined as extensions of procedural languages by data abstractions with inheritance [CW85]. In object-oriented languages programs are arranged as objects. An object contains an internal state together with methods to manipulate it. In class-based languages, like Smalltalk [GR83], C++ [Str86], Eiffel [Mey92] and Java [AG96] among others, classes are used for incremental program construction. Classes can be used to create new objects sharing the implementation common to all objects belonging to the class, its instances, or to incrementally define new classes by inheritance, where parts of the old superclass may be used.

Polymorphism. Polymorphic functions are functions whose operands can have more than one type. There are different kinds of polymorphism. One of them is parametric polymorphism. It is obtained when a function works uniformly on a range of types: these types normally have some common structure. Parametric polymorphism is called so because the uniformity of type structure is normally achieved by type parameters. For object-oriented languages, parametric polymorphism can be used to avoid type-casts resulting in more efficient code.

Another form of polymorphism is the inclusion polymorphism introduced by Cardelli and Wegner in [CW85] to model subtypes and inheritance. The idea of a type being a subtype

from another is useful for expressing that basic types like *Integers* are subtype from *Reals*, written $Int \leq Real$ but also for more complex structures such as a type representing *Toyotas* which is a subtype from a more general type such as *Vehicles* ($Toyota \leq Vehicle$). Every object of a subtype can be used in a supertype context in the sense that every *Toyota* is a *vehicle* and can be operated on by all operations that are applicable to vehicles.

In languages like older versions of Java, the subtyping is only based on inheritance, and it is called nominal subtyping, which means that one type is a subtype of another if and only if it is explicitly declared to be so in its definition. That was one of the major drawbacks in the design of Java: its lack of parametric polymorphism. In the version 1.5 though, Java is being updated to incorporate some new elements in the language: parametrized types and type variables to support the creation of generic code. This development is based on an earlier proposal called Generic Java (GJ) [BSM⁺01].

Subtype-bounded polymorphism. An important contribution for typed object-oriented languages is the combination of parametric polymorphism and subtype polymorphism. There are different systems for combining these two concepts, one of them is Cardelli and Wegner's "Bounded Fun" [CW85], a second-order polymorphic λ -calculus with bounded subtyping and records for modeling object-oriented languages. Another system, F -bounded quantification [BGR99], is the base for the generics in Java.

Let us clarify the concept with an standard example: point objects with an internal state denoting its coordinates and two methods for manipulating it, and a subtype *CPoint* that represents colored points. In Java we would define them like this:

```
class Point {
    ...
    public void setX() {...}
    public int getX() {...}
    ...
}
class CPoint extends Point {...}
```

We can type the invocation of the *getX* method with $\forall A \leq Point. A \rightarrow Int$, where A ranges over all subtypes of *Point*, for example *CPoint*. The structural definition of the subtyping relation guarantees that all elements for all subtypes of *Point* share some common properties, in this example assuring that *CPoint* supports at least a *getX* and a *setX* method.

System F_{\leq}^{ω} . The system that we study in this thesis is one of the simplest systems that combines parametric polymorphism and bounded polymorphism, F_{\leq}^{ω} (called "F-omega-sub"). If we start with the simple typed λ -calculus [Chu40], and we add parametric polymorphism, then we obtain Girard's system F [Gir71]. Adding type operators, i.e. functions from types to types, yields F^{ω} [Gir72], a system with higher order polymorphism. Moreover, extending the system F in another direction, by adding subtyping and bounded quantification, yields system F_{\leq} . The integration of these two systems, F_{\leq} and F^{ω} leads to system F_{\leq}^{ω} .

We analyze in this thesis the metatheory of the system, i.e., subtyping algorithms. We omit the treatment of typechecking since the major difficulties in the metatheory are in the subtyping.

We provide new proofs of soundness, completeness and termination for a known subtyping algorithm. That algorithm was studied by Steffen and Pierce in [PS97], but the proofs they provide are more complicated, based on a strong normalization theorem. The metatheory of a similar system, F_{\leq}^{ω} , was also studied by Compagnoni and Goguen [CG03], [CG99]. They proved the soundness and completeness of the algorithms using Kripke models. Usually for these algorithms, soundness is relatively easy to prove, and completeness is the real issue.

In 2006, Andreas Abel developed an algorithm for a polarized version of $F_{<}^{\omega}$ without bounded quantification and extended it to sized types in [Abe06b], using a very different proof technique for showing completeness of the algorithm, without strong normalization or model construction, but using a lexicographical induction on kinds and derivations. An open question in that paper was whether these much easier proofs extend to the whole system $F_{<}^{\omega}$ with bounded quantification. This question I have investigated in this thesis, and I give a positive answer. We prove the soundness, completeness and termination of the algorithm, using only syntactical proofs. More exactly, I have investigated two questions in this thesis:

1. It is possible to prove the completeness and termination of the subtyping algorithm for the system $F_{<}^{\omega}$ [PS97] using only syntactical means?
2. It is possible to prove the completeness and termination of the subtyping algorithm for the polarized system $F_{<}^{\omega}$, similar to the system studied by Steffen in [Ste98], using only syntactical means?

We could answer only the first question affirmatively. For the second question, we do not have the answer yet. We conjecture it is possible as well though, but it remains an open question.

Contents. This thesis consists of two parts. The first part studies algorithmic subtyping for system $F_{<}^{\omega}$. In this part we follow the approach of fully normalizing constructors, and then defining an algorithm for constructors in normal form. In this approach the correctness and completeness of the normalizer has to be proven as well.

In the second part we study the polarized system $F_{<}^{\omega}$, and its metatheory. In this system the constructors are distinguished with regard to their monotonicity or variance. We do not extend the previous algorithm to the polarized system, but we investigate another approach: weak head normalization of the constructors. The soundness of the algorithm is easy to prove, but its completeness remains open. For this reason, the second part does not contain formal proofs. We describe system and the algorithm, and its implementation in Haskell.

In the first chapter we recapitulate system $F_{<}^{\omega}$ with kinding, equality and subtyping. In the second chapter we use hereditary substitution for the normalization of constructors, and we prove the soundness and completeness of the normalization function. Afterwards, in the third chapter, we define the algorithm for deciding subtyping, which works on terms in normal form and we also show its soundness, completeness, and termination.

We extend system $F_{<}^{\omega}$ with polarities in chapter four, and define again kinding, equality, and subtyping for the new system. In chapter five we define algorithmic subtyping rules for the polarized system, this time using weak head normalization, and we describe its implementation in Haskell. Finally, we review the results of the thesis in the conclusions.

Part I.

Higher Order Subtyping

1. System $F_{<}^\omega$:

This chapter introduces the syntax and the rules of kinding, equality, and subtyping of system $F_{<}^\omega$, a typed λ -calculus of polymorphic functions, subtyping, and type operators. Besides term abstraction $\lambda(x : A) t$ and application $r s$ of the simply typed λ -calculus and type abstraction $\lambda(X : \kappa) t$ and application $t A$ of the second-order polymorphic λ -calculus, it includes functions on the level of kinds, allowing abstraction $\lambda(X : F) G$ and application $F G$ within type expressions. To guarantee the well-formedness of applications within types, an extra level of *kinds* is introduced: the kind $*$ classifies ordinary types (which are inhabited by terms), while kinds of the form $\kappa_1 \rightarrow \kappa_2$ classify type operators: functions mapping types from kind κ_1 to types of kind κ_2 .

The basic judgement of system $F_{<}^\omega$ is the typing judgement $\Gamma \vdash t : A$, read "term t has type A in context Γ ", where Γ records the type of each free term variable and the kind of each type variable. The equivalent judgement for types is the kinding judgement $\Gamma \vdash F : \kappa$, read "type F has kind κ in context Γ ".

To extend system F^ω with subtyping, the declaration of each type variable X in context Γ is extended with an upper bound, written $X \leq G$, which constraints X to range only over subtypes of G in the appropriate kind. To allow new constraints of this form to be introduced in the system, the universal quantifier of system F is extended to a bounded quantifier $\forall X \leq G : \kappa. A$.

To ensure that the new system can still type all the terms of F^ω , we need a maximal element \top_κ in every kind κ . In [PS97], Steffen and Pierce introduce for each kind such a maximal element \top_κ . Instead of that, we introduce only \top of kind $*$ and the rule $A \leq \top : *$, and define $\top_{\vec{\kappa} \rightarrow *} = \lambda \vec{Y} \top$, where the lengths of $\vec{\kappa}$ and \vec{Y} coincide. An easy induction shows that the so defined \top_κ is indeed a maximal element in its kind, i.e. that the rule $F \leq \top_\kappa : \kappa$ is derivable. This way we keep the language simpler. The assumption $X \leq \top_\kappa : \kappa$ replaces $X : \kappa$ in the context.

For kinds of the form $\kappa_1 \rightarrow \kappa_2$ the subtype relation is just the pointwise extension of subtyping for κ_2 : a function $F : \kappa_1 \rightarrow \kappa_2$ is smaller than a function $G : \kappa_1 \rightarrow \kappa_2$ if $FH \leq GH$ for every $H : \kappa_1$. In part II we will see richer definitions of operator subtyping.

At the base kind $*$, the subtype relation also includes rules for the types $A \rightarrow B$ and $\forall X \leq G : \kappa. A$. The rule for arrow types is the following:

$$\frac{\Gamma \vdash B_1 \leq A_1 : * \quad \Gamma \vdash A_2 \leq B_2 : *}{\Gamma \vdash A_1 \rightarrow A_2 \leq B_1 \rightarrow B_2 : *} \quad (\text{S-ARROW})$$

The direction of the subtype relation is reversed (contravariant) for the argument types in the left-hand premise, while it runs in the same direction (covariant) for the result types as for the function themselves. The intuition is that, if we have a function f of type $A_1 \rightarrow A_2$, then we know that f accepts elements of type A_1 ; clearly, f will also accept elements of any subtype B_1 of A_1 . The type of f also tells us that it returns elements of type A_2 ; we can also view these results belonging to any supertype B_2 of A_2 . That is, any function f of type $A_1 \rightarrow A_2$ can also be viewed as having type $B_1 \rightarrow B_2$.

The subtyping rule for bounded quantifiers is equally simple:

$$\frac{\Gamma, X \leq G:\kappa \vdash A_1 \leq A_2: *}{\Gamma \vdash \forall X \leq G:\kappa. A_1 \leq \forall X \leq G:\kappa. A_2} \quad (\text{S-ALL})$$

That is, a polymorphic value $f \in \forall X \leq G:\kappa. A_1$ can be used in a context that expects an element of $\forall X \leq G:\kappa. A_2$, provided that, for each legal argument type F , the value of f at F can safely be used as an element of A_2 . This rule is called "kernel" rule, because the bounds of the two quantifiers being compared must be equal. The term "kernel" comes from Cardelli and Wegner's paper [CW85], where this variant of the system is called Kernel Fun.

It would be semantically sensible to refine the left-hand premise of this rule so that it only requires $G' \leq G$ where X is constrained to the *common* part of their domain:

$$\frac{\Gamma \vdash G' \leq G:\kappa \quad \Gamma, X \leq G' \vdash A_1 \leq A_2: *}{\Gamma \vdash \forall X \leq G:\kappa. A_1 \leq \forall X \leq G':\kappa. A_2: *}$$

The extra flexibility offered by this refinement is very costly: this rule is responsible for the failure of a number of important proof-theoretic properties in standard formulations of F_{\leq}^{ω} , including decidability of subtyping [Pie92].

1.1. Constructors and kinds

The System F_{\leq}^{ω} consists of terms, type constructors and kinds. We do not consider terms because we investigate only subtyping and not typing. Constructors are classified by their kind, i.e., as types, functions on types, etc.

Kinds.

$$\kappa ::= * \mid \kappa_1 \rightarrow \kappa_2$$

Let $\vec{\kappa} \rightarrow \kappa'$ be an abbreviation for $\kappa_1 \rightarrow \kappa_2 \rightarrow \dots \rightarrow \kappa_n \rightarrow \kappa'$ where $|\vec{\kappa}| = n$. Let $|\kappa| \in \mathbb{N}$ denote a measure on kinds with $|\kappa'| \leq |\kappa \rightarrow \kappa'|$ and $|\kappa| < |\kappa \rightarrow \kappa'|$. An example of such a measure is the *rank*. The rank of a kind is defined recursively as $\text{rk}(\vec{\kappa} \rightarrow *) = \max\{1 + \text{rk}(\kappa_i) \mid 1 \leq i \leq |\vec{\kappa}|\}$ (where the maximum of an empty set is 0).

Constructors are given by the following Curry-style type-level λ -calculus. The meta-variable X ranges over a countable infinite set of constructor variables.

Constructors.

$$A, B, F, G ::= X \mid \lambda X F \mid F G \mid A \rightarrow B \mid \forall X \leq G:\kappa. A \mid \top$$

As usual, $\lambda X F$ binds variable X in F . We identify constructors up to α -equivalence, i.e., up to renaming of bound variables.

We use U, V, W for β -normal constructors. We use A, B for constructors of kind $*$ (types).

Free Variables $\text{FV}(F)$. We define the set $\text{FV}(F)$ of variables occurring free in F inductively as follows:

$$\begin{aligned}
\text{FV}(X) &:= X \\
\text{FV}(\lambda XF) &:= \text{FV}(F) \setminus \{X\} \\
\text{FV}(FG) &:= \text{FV}(F) \cup \text{FV}(G) \\
\text{FV}(A \rightarrow B) &:= \text{FV}(A) \cup \text{FV}(B) \\
\text{FV}(\forall X \leq G : \kappa. A) &:= \text{FV}(G) \cup (\text{FV}(A) \setminus \{X\}) \\
\text{FV}(\top) &:= \emptyset
\end{aligned}$$

Free Variables $\text{FV}(\Gamma)$. We define the set $\text{FV}(\Gamma)$ of variables occurring free in the bounds in Γ inductively as follows:

$$\begin{aligned}
\text{FV}(\diamond) &:= \emptyset \\
\text{FV}(\Gamma, X \leq G : \kappa) &:= \text{FV}(\Gamma) \cup \text{FV}(G)
\end{aligned}$$

Substitution $[G/X]F$. We define the result $[G/X]F$ of replacing every free occurrence of the variable X in F by the term G inductively as follows:

$$\begin{aligned}
[G/X]Y &:= \begin{array}{ll} G & \text{if } X = Y \\ Y & \text{otherwise} \end{array} \\
[G/X](\lambda Y F) &:= \lambda Y. [G/X]F \quad \text{where we assume by renaming of } Y \\
&\quad \text{that } Y \notin \{X\} \cup \text{FV}(G) \\
[G/X](FH) &:= ([G/X]F) ([G/X]H) \\
[G/X](A \rightarrow B) &:= [G/X]A \rightarrow [G/X]B \\
[G/X](\forall Y \leq H : \kappa. A) &:= \forall Y \leq [G/X]H : \kappa. [G/X]A \quad \text{where we assume by renaming of } Y \\
&\quad \text{that } Y \notin \{X\} \cup \text{FV}(G) \\
[G/X]\top &:= \top
\end{aligned}$$

Contexts follow the grammar $\Gamma ::= \diamond \mid X \leq G : \kappa$. We define the notation $\Gamma, X : \kappa$ as an abbreviation for $\Gamma, X \leq \top : \kappa$. We assume all bound variables in Γ to be distinct.

Substitution can be extended to contexts by substituting in every bound of the context.

Substitution $[G/X]\Gamma$.

$$\begin{aligned}
[G/X]\diamond &= \diamond \\
[G/X](\Gamma, Y \leq H : \kappa) &= [G/X]\Gamma, Y \leq [G/X]H : \kappa
\end{aligned}$$

1.2. Kinding and wellformed contexts

Well-formed contexts. Well-formed contexts $\Gamma \vdash$ are constructed from the empty context by adding well-kinded type variable declarations. They are defined inductively as follows, mutually with the kinding judgement $\Gamma \vdash F : \kappa$.

$$\boxed{\Gamma \vdash}$$

$$\frac{}{\diamond \vdash} \text{ (C-EMPTY)} \quad \frac{\Gamma \vdash \quad \Gamma \vdash G : \kappa}{\Gamma, X \leq G : \kappa \vdash} \text{ (C-BOUND)}$$

The rules (K-ABS), (K-APP) and (K-ARROW) are as for system F^ω , rule (K-ALL) inserts a new bound in the context. The new rules are: (K-TOP): \top is a type in a well-formed context and (K-VAR-BOUND): bounded variables from the context can be typed if their bounds can. This is a very important feature in order to ensure termination of the subtyping algorithm that will be explained later on. We maintain the invariant that kinding statements are only derivable in well-formed contexts. This is the validity of kinding, that will be proved in the following. (Lemma 1.2.2).

$$\begin{array}{c}
\text{Kinding} \quad \boxed{\Gamma \vdash F : \kappa} \\
\\
\frac{(X \leq G : \kappa) \in \Gamma \quad \Gamma \vdash G : \kappa}{\Gamma \vdash X : \kappa} \text{ (K-VAR-BOUND)} \quad \frac{\Gamma \vdash}{\Gamma \vdash \top : *} \text{ (K-TOP)} \\
\\
\frac{\Gamma, X : \kappa \vdash F : \kappa'}{\Gamma \vdash \lambda X F : \kappa \rightarrow \kappa'} \text{ (K-ABS)} \quad \frac{\Gamma \vdash F : \kappa \rightarrow \kappa' \quad \Gamma \vdash G : \kappa}{\Gamma \vdash F G : \kappa'} \text{ (K-APP)} \\
\\
\frac{\Gamma \vdash A : * \quad \Gamma \vdash B : *}{\Gamma \vdash A \rightarrow B : *} \text{ (K-ARROW)} \quad \frac{\Gamma, X \leq G : \kappa \vdash A : *}{\Gamma \vdash \forall X \leq G : \kappa. A : *} \text{ (K-ALL)}
\end{array}$$

Lemma 1.2.1 (Inversion of kinding)

1. If $\Gamma \vdash X : \kappa$ then $(X \leq G : \kappa) \in \Gamma$ and $\Gamma \vdash G : \kappa$ for some G .
2. If $\Gamma \vdash \lambda X F : \kappa \rightarrow \kappa'$ then $\Gamma, X : \kappa \vdash F : \kappa'$
3. If $\Gamma \vdash A \rightarrow B : *$ then $\Gamma \vdash A : *$ and $\Gamma \vdash B : *$.
4. If $\Gamma \vdash \forall X \leq G : \kappa. A : *$ then $\Gamma, X \leq G : \kappa \vdash A : *$.
5. If $\Gamma \vdash F G : \kappa'$ then $\Gamma \vdash F : \kappa \rightarrow \kappa'$ and $\Gamma \vdash G : \kappa$ for some κ .
6. If $\mathcal{D} :: \Gamma \vdash F \vec{G} : \kappa'$ then $\mathcal{D}' :: \Gamma \vdash F : \vec{\kappa} \rightarrow \kappa'$ and $\mathcal{D}_i :: \Gamma \vdash G_i : \kappa_i$ for some $\vec{\kappa}$, and $\mathcal{D}', \mathcal{D}_i < \mathcal{D}$.

Proof. Propositions (1) until (5) are trivial. Proposition (6) is proven by induction on $|\vec{G}|$.

Case $|\vec{G}| = 0$. Nothing to show.

Case $|\vec{G}| = n$. We have $\Gamma \vdash F \vec{G} : \kappa'$ and we show $\Gamma \vdash F : \vec{\kappa} \rightarrow \kappa'$ and $\Gamma \vdash G_i : \kappa_i$. By inversion, part (5) we get $\mathcal{D}'' :: \Gamma \vdash F G_1 \dots G_{n-1} : \kappa_n \rightarrow \kappa'$ and $\mathcal{D}_n :: \Gamma \vdash G_n : \kappa_n$. Then we can apply the induction hypothesis and get $\mathcal{D}' :: \Gamma \vdash F : \vec{\kappa} \rightarrow \kappa'$ and $\mathcal{D}_i :: \Gamma \vdash G_i : \kappa_i$ for $1 \leq i \leq n-1$, which, together with \mathcal{D}_n , finishes the proof. \square

Lemma 1.2.2 (Admissible rules for kinding)

1. *Validity:* If $\mathcal{D} :: \Gamma \vdash F : \kappa$ then $\Gamma \vdash$.

2. *Weakening:* If $\mathcal{D} :: \Gamma, \Gamma' \vdash F : \kappa$ and $\Gamma \vdash G : \kappa$ then $\Gamma, X \leq G : \kappa, \Gamma' \vdash F : \kappa$.
3. *Strengthening:* If $\mathcal{D} :: \Gamma, X \leq G : \kappa, \Gamma' \vdash F : \kappa'$ and $X \notin \text{FV}(F, \Gamma')$, then $\mathcal{D}' :: \Gamma, \Gamma' \vdash F : \kappa'$.
4. *Substitution:* If $\mathcal{D} :: \Gamma, X \leq H : \kappa, \Gamma' \vdash F : \kappa'$ and $\Gamma \vdash G : \kappa$ then $\Gamma, [G/X]\Gamma' \vdash [G/X]F : \kappa'$.

Proof. Each by induction on \mathcal{D} . □

This lemma captures the intuition behind well-formed contexts: when there is a bound G in a well-formed context, then G is well-kinded in that context.

Lemma 1.2.3 *If $\mathcal{D} :: \Gamma \vdash$ and $X \leq G : \kappa \in \Gamma$ then $\Gamma \vdash G : \kappa$.*

Proof. By induction on \mathcal{D} .

Case (C-EMPTY) Impossible.

Case (C-BOUND) We have $\Gamma, X' \leq G' : \kappa' \vdash$. If $X = X'$ (implying $G = G'$) we have by assumption $\Gamma \vdash G : \kappa$ and we are done. Otherwise by induction hypothesis $\Gamma \vdash G : \kappa$ so we are done by weakening. □

The following lemma is a strengthening of Lemma 1.2.3. We will need it in some induction proofs in order to use the induction hypothesis on the bound G .

Lemma 1.2.4 *If $\mathcal{D} :: \Gamma, X \leq G : \kappa, \Gamma' \vdash \mathcal{J}$ then $\mathcal{E} :: \Gamma \vdash G : \kappa$ and $\mathcal{E} \leq \mathcal{D}$.*

By $\mathcal{D} :: \Gamma \vdash \mathcal{J}$ we mean \mathcal{D} is a derivation (proof tree) of judgement $\Gamma \vdash \mathcal{J}$. $\mathcal{E} \leq \mathcal{D}$ means \mathcal{E} is a derivation of smaller height than \mathcal{D} .

1.3. Similarity of contexts

Other than in System F^ω we have bounds in the context. But by looking at the rules for kinding, and later the rules for equality, we see that the bounds are not important for kinding or equality. They are only relevant for subtyping, which we will see later. For this reason we define a notion of similarity of contexts. These are contexts that are not identical, because their variables might have different bounds, but they contain the same variables and assign the same kinds to them. This makes them identical if we ignore the bounds, and our goal will be to prove that wellkinded constructors in one context, are also wellkinded in a similar context, and the same for equality.

Similar contexts $\Gamma \simeq \Gamma'$

$$\Gamma \simeq \Gamma' \iff \forall X. \forall \kappa. (\exists G. (X \leq G : \kappa) \in \Gamma) \iff (\exists G'. (X \leq G' : \kappa) \in \Gamma')$$

Lemma 1.3.1 *If $\mathcal{D} :: \Gamma \vdash F : \kappa$ and $\Gamma' \vdash$ and $\Gamma \simeq \Gamma'$ then $\Gamma' \vdash F : \kappa$.*

Proof.

By induction on \mathcal{D} .

Case (K-VAR-BOUND) We have $\Gamma \vdash X : \kappa$ and $(X \leq G : \kappa) \in \Gamma$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash X : \kappa$. $\Gamma \simeq \Gamma'$ implies that there is a G' with $(X \leq G' : \kappa) \in \Gamma'$, and because $\Gamma' \vdash$ holds, by lemma 1.2.3 $\Gamma' \vdash G' : \kappa$. Thus, we can finish by rule (K-VAR-BOUND).

Case (K-TOP) We have $\Gamma \vdash \top : *$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash \top : *$. By assumption $\Gamma' \vdash$, thus, we can finish with (K-TOP).

Case (K-ABS) We have $\Gamma \vdash \lambda XF : \kappa \rightarrow \kappa'$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash \lambda XF : \kappa$. Since $\Gamma, X : \kappa \simeq \Gamma', X : \kappa$. By induction hypothesis we have $\Gamma', X : \kappa \vdash F : \kappa'$, thus, we can finish by (K-ABS).

Case (K-APP) We have $\Gamma \vdash FG : \kappa'$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash FG : \kappa'$. By induction hypothesis we have $\Gamma' \vdash F : \kappa \rightarrow \kappa'$ and $\Gamma' \vdash G : \kappa$, thus, we can finish with (K-APP).

Case (K-ARROW) We have $\Gamma \vdash A \rightarrow B : *$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash A \rightarrow B : *$. By induction hypothesis we have $\Gamma' \vdash A : *$ and $\Gamma' \vdash B : *$, thus, we can finish with (K-ARROW).

Case (K-ALL) We have $\Gamma \vdash \forall X \leq G : \kappa. A : *$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash \forall X \leq G : \kappa. A : *$. If $\Gamma \simeq \Gamma'$ then also $\Gamma, X \leq G : \kappa \simeq \Gamma', X \leq G : \kappa$. Thus, we have by induction hypothesis $\Gamma', X \leq G : \kappa \vdash A : *$. Thus, we can finish by (K-ALL). □

1.4. Equality

In contrast to most presentations of System F^ω , we consider constructors equivalent modulo β and η . Beta reduction expresses the idea of function application. The beta reduct of $((\lambda X.F)G)$ is simply $[G/X]F$. Eta conversion expresses the idea of extensionality, which in this context means that two functions are the same if and only if they give the same result for all arguments. Eta converts between $\lambda X.FX$ and F whenever X does not appear free in F .

The $\beta\eta$ -equality is the reflexiv-symmetric-transitiv closure of the congruent closure of the $\beta\eta$ -relation given by the axioms (EQ- β) and (EQ- η).

$$\boxed{\Gamma \vdash F = F' : \kappa}$$

Axioms.

$$\frac{\Gamma, X : \kappa \vdash F : \kappa' \quad \Gamma \vdash G : \kappa}{\Gamma \vdash (\lambda XF)G = [G/X]F : \kappa'} \text{ (EQ-}\beta\text{)} \quad \frac{\Gamma \vdash F : \kappa \rightarrow \kappa'}{\Gamma \vdash (\lambda X.FX) = F : \kappa \rightarrow \kappa'} X \notin \text{FV}(F) \text{ (EQ-}\eta\text{)}$$

Congruence rules.

$$\begin{array}{c}
\frac{(X \leq G : \kappa) \in \Gamma \quad \Gamma \vdash G : \kappa}{\Gamma \vdash X = X : \kappa} \text{ (EQ-VAR-BOUND)} \quad \frac{\Gamma \vdash}{\Gamma \vdash \top = \top : *} \text{ (EQ-TOP)} \\
\\
\frac{\Gamma, X : \kappa \vdash F = F' : \kappa'}{\Gamma \vdash \lambda X F = \lambda X F' : \kappa \rightarrow \kappa'} \text{ (EQ-}\lambda\text{)} \quad \frac{\Gamma \vdash G = G' : \kappa \quad \Gamma, X \leq G : \kappa \vdash A = A' : *}{\Gamma \vdash \forall X \leq G : \kappa. A = \forall X \leq G' : \kappa. A' : *} \text{ (EQ-}\forall\text{)} \\
\\
\frac{\Gamma \vdash F = F' : \kappa \rightarrow \kappa' \quad \Gamma \vdash G = G' : \kappa}{\Gamma \vdash F G = F' G' : \kappa'} \text{ (EQ-APP)} \\
\\
\frac{\Gamma \vdash A = A' : * \quad \Gamma \vdash B = B' : *}{\Gamma \vdash A \rightarrow B = A' \rightarrow B' : *} \text{ (EQ-ARROW)}
\end{array}$$

Symmetry and transitivity.

$$\frac{\Gamma \vdash F = F' : \kappa}{\Gamma \vdash F' = F : \kappa} \text{ (EQ-SYM)} \quad \frac{\Gamma \vdash F_1 = F_2 : \kappa \quad \Gamma \vdash F_2 = F_3 : \kappa}{\Gamma \vdash F_1 = F_3 : \kappa} \text{ (EQ-TRANS)}$$

Reflexivity is not directly given as a rule, but it can be easily proved. The other admissible rules are extensions of the standard rules.

Admissible rules for equality

$$\begin{array}{c}
\frac{\Gamma \vdash F : \kappa}{\Gamma \vdash F = F : \kappa} \text{ (EQ-REFL)} \quad \frac{\Gamma \vdash F : \vec{\kappa} \rightarrow *}{\Gamma \vdash F = \lambda Y_1 \dots \lambda Y_n. F Y_1 \dots Y_n : \vec{\kappa} \rightarrow *} \text{ (EQ-}\eta\text{-VECT)} \\
\\
\frac{\Gamma, \vec{Y} : \vec{\kappa} \vdash F = F' : \kappa'}{\Gamma \vdash \lambda \vec{Y}. F = \lambda \vec{Y}. F' : \vec{\kappa} \rightarrow \kappa'} \text{ (EQ-}\lambda\text{-VECT)} \\
\\
\frac{\Gamma \vdash F = F' : \vec{\kappa} \rightarrow \kappa' \quad \Gamma \vdash G_i = G'_i : \kappa_i \text{ for all } i}{\Gamma \vdash F \vec{G} = F' \vec{G}' : \kappa'} \text{ (EQ-APP-VECT)}
\end{array}$$

Proof. of rule (EQ- λ -VECT) by induction on $|\vec{\kappa}|$.

Case $|\vec{\kappa}| = 0$. Nothing to show.

Case $|\vec{\kappa}| = n$. We have $\Gamma, \vec{Y} : \vec{\kappa} \vdash F = F' : \kappa'$ and we show $\Gamma \vdash \lambda \vec{Y}. F = \lambda \vec{Y}. F' : \vec{\kappa} \rightarrow \kappa'$.

We have by induction hypothesis $\Gamma, Y_1 : \kappa_1 \vdash \lambda Y_2 \dots \lambda Y_n. F = \lambda Y_2 \dots \lambda Y_n. F' : \kappa_2 \rightarrow \dots \rightarrow \kappa_n \rightarrow \kappa'$ so we can apply (EQ- λ):

$$\frac{\Gamma, Y_1 : \kappa_1 \vdash \lambda Y_2 \dots \lambda Y_n. F = \lambda Y_2 \dots \lambda Y_n. F' : \kappa_2 \rightarrow \dots \rightarrow \kappa_n \rightarrow \kappa'}{\Gamma \vdash \lambda \vec{Y}. F = \lambda \vec{Y}. F' : \vec{\kappa} \rightarrow \kappa'} \text{ (EQ-}\lambda\text{)}$$

□

Proof. of rule (EQ-APP-VECT) by induction on $|\vec{\kappa}|$.

Case $|\vec{\kappa}| = 0$. Nothing to show.

Case $|\vec{\kappa}| = n$. We have $\Gamma \vdash F = F' : \vec{\kappa} \rightarrow \kappa'$ and $\Gamma \vdash G_i = G'_i : \kappa_i$ and we show $\Gamma \vdash F \vec{G} = F' \vec{G}' : \kappa'$.

We have by induction hypothesis $\Gamma \vdash F G_1 \dots G_{n-1} = F' G'_1 \dots G'_{n-1} : \kappa_n \rightarrow \kappa'$ so we can apply (EQ-APP):

$$\frac{\Gamma \vdash F G_1 \dots G_{n-1} = F' G'_1 \dots G'_{n-1} : \kappa_n \rightarrow \kappa' \quad \Gamma \vdash G_n = G'_n : \kappa_n}{\Gamma \vdash F \vec{G} = F' \vec{G}' : \kappa'} \text{ (EQ-APP)}$$

□

Proof. of rule (EQ- η -VECT) by induction on $|\vec{\kappa}|$.

Case $|\vec{\kappa}| = 0$. Nothing to show.

Case $|\vec{\kappa}| = n$. We have $\Gamma \vdash F : \vec{\kappa} \rightarrow *$ and we show $\Gamma \vdash F = \lambda Y_1 \dots \lambda Y_n. F Y_1 \dots Y_n : \vec{\kappa} \rightarrow *$. We have by induction hypothesis $\Gamma \vdash F = \lambda Y_1 \dots \lambda Y_{n-1}. F Y_1 \dots Y_{n-1} : \vec{\kappa} \rightarrow *$ thus we can derive:

$$\frac{\Gamma, \vec{Y} : \vec{\kappa} \vdash F Y_1 \dots Y_{n-1} : \kappa_n \rightarrow *}{\Gamma, Y_1, \dots, Y_{n-1} \vdash F Y_1 \dots Y_{n-1} = \lambda Y_n. F \vec{Y}} \text{ (EQ-}\eta\text{)}$$

$$\frac{\Gamma \vdash F = \lambda Y_1 \dots \lambda Y_{n-1}. F Y_1 \dots Y_{n-1} \quad \Gamma \vdash \lambda Y_1 \dots Y_{n-1}. F Y_1 \dots Y_{n-1} = \lambda \vec{Y}. F \vec{Y}}{\Gamma \vdash F = \lambda \vec{Y}. F \vec{Y}}$$

□

Lemma 1.4.1 (Admissible rules for equality II)

1. *Weakening:* If $\mathcal{D} :: \Gamma, \Gamma' \vdash F = F' : \kappa$ and $\Gamma \vdash G : \kappa$ then $\Gamma, X \leq G : \kappa, \Gamma' \vdash F = F' : \kappa$.
2. *Strengthening:* If $\mathcal{D} :: \Gamma, X \leq G, \Gamma' : \kappa \vdash F = F' : \kappa'$ and $X \notin \text{FV}(F, F', \Gamma')$, then $\mathcal{D}' :: \Gamma, \Gamma' \vdash F = F' : \kappa'$.
3. *Substitution:* If $\mathcal{D} :: \Gamma, X \leq H : \kappa, \Gamma' \vdash F = F' : \kappa'$ and $\Gamma \vdash G : \kappa$ then $\Gamma, [G/X]\Gamma' \vdash [G/X]F = [G/X]F' : \kappa'$.

Proof. Each by induction on \mathcal{D} .

□

If two constructors are declared as equal in a given context, then they can be also declared as equal in a similar context. The proof is a trivial induction.

Lemma 1.4.2 *If* $\mathcal{D} :: \Gamma \vdash G = G' : \kappa$ *and* $\Gamma' \vdash$ *and* $\Gamma \simeq \Gamma'$ *then* $\Gamma' \vdash G = G' : \kappa$.

Proof.

By induction on \mathcal{D} .

Case (EQ- β) We have $\Gamma \vdash (\lambda X. F X) = [G/X]F : \kappa \rightarrow \kappa'$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash (\lambda X. F X) = [G/X]F : \kappa \rightarrow \kappa'$. We have by induction hypothesis $\Gamma', X : \kappa \vdash F : \kappa'$ and $\Gamma' \vdash G : \kappa$. Thus we can finish by (EQ- β).

- Case* (EQ- η) We have $\Gamma \vdash (\lambda X.FX) = F : \kappa \rightarrow \kappa'$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash (\lambda X.FX) = F : \kappa \rightarrow \kappa'$. By induction hypothesis we have $\Gamma' \vdash F : \kappa \rightarrow \kappa'$ thus we can finish by (EQ- η).
- Case* (EQ-VAR-BOUND) We have $\Gamma \vdash X = X : \kappa$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash X = X : \kappa$. Since $\Gamma \simeq \Gamma'$ and $(X \leq G : \kappa) \in \Gamma$ there is a G' with $(X \leq G' : \kappa) \in \Gamma'$ and because $\Gamma' \vdash$ by assumption we have $\Gamma' \vdash G' : \kappa$ by lemma 1.2.3. Thus we can finish with (EQ-VAR-BOUND).
- Case* (EQ-TOP) We have $\Gamma \vdash \top : *$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash \top : *$. Since we have $\Gamma' \vdash$ by assumption we can apply (EQ-TOP) and we are done.
- Case* (EQ- λ) We have $\Gamma \vdash \lambda XF = \lambda XF' : \kappa \rightarrow \kappa'$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash \lambda XF = \lambda XF' : \kappa \rightarrow \kappa'$. Since $\Gamma \simeq \Gamma'$ holds $\Gamma, X : \kappa \simeq \Gamma', X' : \kappa$ thus, by induction hypothesis we have $\Gamma', X : \kappa \vdash F = F' : \kappa'$ thus we can finish with (EQ- λ).
- Case* (EQ- \forall) We have $\Gamma \vdash \forall X \leq G : \kappa. A : * = \forall X \leq G' : \kappa. A' : *$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash \forall X \leq G : \kappa. A : * = \forall X \leq G' : \kappa. A' : *$. Since $\Gamma \simeq \Gamma'$ also holds $\Gamma, X \leq G : \kappa \simeq \Gamma', X \leq G' : \kappa$ thus we get the induction hypothesis $\Gamma', X \leq G : \kappa \vdash A = A' : \kappa$ and we also get by induction hypothesis directly $\Gamma' \vdash G = G' : \kappa$. Thus we can apply (EQ- \forall) and we are done.
- Case* (EQ-APP) We have $\Gamma \vdash FG = F'G' : \kappa'$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash FG = F'G' : \kappa'$. By induction hypothesis we have $\Gamma' \vdash F : \kappa \rightarrow \kappa'$ and $\Gamma' \vdash G : \kappa$. Thus we finish by (EQ-APP).
- Case* (EQ-ARROW) We have $\Gamma \vdash A \rightarrow B = A' \rightarrow B' : *$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash A \rightarrow B = A' \rightarrow B' : *$. By induction hypothesis we have $\Gamma' \vdash A = A' : *$ and $\Gamma' \vdash B = B' : *$. Thus we finish with (EQ-ARROW).
- Case* (EQ-SYM) We have $\Gamma \vdash F' = F : \kappa$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash F' = F : \kappa$. By induction hypothesis we have $\Gamma' \vdash F = F' : \kappa$ thus we can finish with (EQ-SYM).
- Case* (EQ-TRANS) We have $\Gamma \vdash F_1 = F_3 : \kappa$ and $\Gamma \simeq \Gamma'$ and we show $\Gamma' \vdash F_1 = F_3 : \kappa$. We have by induction hypothesis $\Gamma' \vdash F_1 = F_2 : \kappa$ and $\Gamma' \vdash F_2 = F_3 : \kappa$ thus we can finish with (EQ-TRANS). □

It is important to show the validity of equality, i.e., if we declare two constructors as equal in a context, we can also kind the constructors in the same context, and they have the same kind.

Lemma 1.4.3 (Validity of equality) *If $\mathcal{D} :: \Gamma \vdash G = G' : \kappa$ then $\Gamma \vdash G : \kappa$ and $\Gamma \vdash G' : \kappa$.*

Proof. By induction on \mathcal{D} . The interesting case is (EQ- \forall).

- Case* (EQ- β) We have $\Gamma \vdash (\lambda XF)G = [G/X]F : \kappa'$ and we show $\Gamma \vdash (\lambda XF)G : \kappa'$ and $\Gamma \vdash [G/X]F : \kappa'$. We have $\Gamma, X : \kappa \vdash F : \kappa'$, so by rule (K-ABS) we get $\Gamma \vdash (\lambda XF)G : \kappa'$ and by lemma 1.2.2, part (4), (substitution) we get $\Gamma \vdash [G/X]F : \kappa'$.

Case (EQ- η) We have $\Gamma \vdash (\lambda X.FX) = F : \kappa \rightarrow \kappa'$ and we show $\Gamma \vdash (\lambda X.FX) : \kappa \rightarrow \kappa'$ and $\Gamma \vdash F : \kappa \rightarrow \kappa'$. We have the second already by assumption. For the first we can derive:

$$\frac{\frac{\Gamma \vdash F : \kappa \rightarrow \kappa'}{\Gamma, X : \kappa \vdash F : \kappa \rightarrow \kappa'} \text{ Weakening} \quad \Gamma, X : \kappa \vdash X : \kappa}{\Gamma, X : \kappa \vdash FX : \kappa'} \text{ (K-APP)} \quad \frac{\Gamma, X : \kappa \vdash FX : \kappa'}{\Gamma \vdash \lambda X.FX : \kappa \rightarrow \kappa'} \text{ (K-ABS)}$$

Case (EQ-VAR-BOUND) We have $\Gamma \vdash X = X : \kappa$ and we show $\Gamma \vdash X : \kappa$. By assumption $X \leq G \in \Gamma$ and $\Gamma \vdash G : \kappa$, so we finish with (K-VAR-BOUND).

Case (EQ-TOP) We have $\Gamma \vdash \top = \top : *$ and we show $\Gamma \vdash \top : *$. By assumption $\Gamma \vdash$ so we finish by (K-TOP).

Case (EQ- λ) We have $\Gamma \vdash \lambda XF = \lambda XF' : \kappa \rightarrow \kappa'$ and we show $\Gamma \vdash \lambda XF : \kappa \rightarrow \kappa'$ and $\Gamma \vdash \lambda XF' : \kappa \rightarrow \kappa'$. By assumption we have $\Gamma, X : \kappa \vdash F : \kappa'$ and $\Gamma, X : \kappa \vdash F' : \kappa'$, so we apply in both cases (K-ABS) to finish.

Case (EQ- \forall) We have $\Gamma \vdash \forall X \leq G : \kappa. A = \forall X \leq G' : \kappa. A' : *$ and we show $\Gamma \vdash \forall X \leq G : \kappa. A : *$ and $\Gamma \vdash \forall X \leq G' : \kappa. A' : *$. We have by induction hypothesis $\Gamma \vdash G : \kappa$ and $\Gamma, X \leq G : \kappa \vdash A : *$ and $\Gamma \vdash G' : \kappa$ and $\Gamma, X \leq G' : \kappa \vdash A' : *$. In the first case, we apply (K-ALL) and we are done. In the second case we have $\Gamma, X \leq G : \kappa \vdash A' : *$ but what we actually need is $\Gamma, X \leq G' : \kappa \vdash A' : *$. But, since $\Gamma, X \leq G : \kappa \simeq \Gamma, X \leq G' : \kappa$ we also get $\Gamma, X \leq G' : \kappa \vdash A' : *$ by lemma 1.3.1, then we finish again with (K-ALL).

Case (EQ-APP) We have $\Gamma \vdash FG = F'G' : \kappa'$ and we show $\Gamma \vdash FG : \kappa'$ and $\Gamma \vdash F'G' : \kappa'$. By induction hypothesis we have $\Gamma \vdash F : \kappa \rightarrow \kappa'$ and $\Gamma \vdash G : \kappa$, $\Gamma \vdash F' : \kappa \rightarrow \kappa'$ and $\Gamma \vdash G' : \kappa$. So we apply in both cases (K-APP).

Case (EQ-ARROW) We have $\Gamma \vdash A \rightarrow B = A' \rightarrow B' : *$ and we show $\Gamma \vdash A \rightarrow B : *$ and $\Gamma \vdash A' \rightarrow B' : *$. By induction hypothesis we have $\Gamma \vdash A : *$, $\Gamma \vdash B : *$, $\Gamma \vdash A' : *$ and $\Gamma \vdash B' : *$. So we apply in both cases the rule (K-ARROW) and we are done.

Case (EQ-SYM) We have $\Gamma \vdash F' = F : \kappa$ and we show $\Gamma \vdash F' : \kappa$ and $\Gamma \vdash F : \kappa$. We are done by the induction hypothesis.

Case (EQ-TRANS) We have $\Gamma \vdash F_1 = F_3 : \kappa$ and we show $\Gamma \vdash F_1 : \kappa$ and $\Gamma \vdash F_3 : \kappa$. We are again done by the induction hypothesis. □

1.5. Subtyping

The F_{\leq}^{ω} subtyping relation $\Gamma \vdash F \leq G : \kappa$ extends the subtyping relation of system F_{\leq} [CW85]. $\beta\eta$ -equivalent types are subtypes of each other (S-EQ), and the subtype relation at every kind is transitive (S-TRANS). Type assumptions from the context may be used as axioms (S-VAR). \top is maximal in the ordering for kind $*$. Subtyping for type operators of higher kind is defined pointwise ((S-ABS), (S-APP)). Arrow- and All-types have the rules discussed above (S-ARROW) and (S-ALL).

Subtyping for higher-order bounded quantification (F_{\leq}^{ω}) $\boxed{\Gamma \vdash F \leq F' : \kappa}$

$$\begin{array}{c} \frac{\Gamma \vdash X \leq G : \kappa \in \Gamma}{\Gamma \vdash X \leq G : \kappa} \text{ (S-VAR)} \quad \frac{\Gamma \vdash F \leq F' : \kappa \rightarrow \kappa' \quad \Gamma \vdash H : \kappa}{\Gamma \vdash FH \leq F'H : \kappa'} \text{ (S-APP)} \\ \\ \frac{\Gamma, X : \kappa \vdash F \leq F' : \kappa'}{\Gamma \vdash \lambda X.F \leq \lambda X.F' : \kappa \rightarrow \kappa'} \text{ (S-ABS)} \quad \frac{\Gamma \vdash A' \leq A : * \quad \Gamma \vdash B \leq B' : *}{\Gamma \vdash A \rightarrow B \leq A' \rightarrow B' : *} \text{ (S-ARROW)} \\ \\ \frac{\Gamma \vdash G : \kappa \quad \Gamma, X \leq G : \kappa \vdash A \leq A' : *}{(\Gamma \vdash \forall X \leq G : \kappa.A) \leq (\forall X \leq G : \kappa.A' : *)} \text{ (S-ALL)} \quad \frac{\Gamma \vdash A : *}{\Gamma \vdash A \leq \top : *} \text{ (S-TOP)} \\ \\ \frac{\Gamma \vdash F = G : \kappa}{\Gamma \vdash F \leq G : \kappa} \text{ (S-EQ)} \quad \frac{\Gamma \vdash F \leq G : \kappa \quad \Gamma \vdash G \leq H : \kappa}{\Gamma \vdash F \leq H : \kappa} \text{ (S-TRANS)} \end{array}$$

Admissible rules for subtyping

$$\begin{array}{c} \frac{\Gamma, (X_i : \kappa_i)_i \vdash W \leq W' : \kappa'}{\Gamma \vdash \lambda \vec{X}.W \leq \lambda \vec{X}.W' : \vec{\kappa} \rightarrow \kappa'} \text{ (S-ABS-VECT)} \\ \\ \frac{\Gamma \vdash F \leq F' : \vec{\kappa} \rightarrow \kappa' \quad \Gamma \vdash G_i : \kappa_i \text{ for all } i}{\Gamma \vdash F\vec{G} \leq F'\vec{G} : \kappa'} \text{ (S-APP-VECT)} \end{array}$$

Proof.

of rule (S-ABS-VECT) by induction on $|\vec{X}|$.

Case $|\vec{X}| = 0$. Nothing to show.

Case $|\vec{X}| = n$. We have $\Gamma, (X_i : \kappa_i)_i \vdash W \leq W' : \kappa'$ and we show $\Gamma \vdash \lambda \vec{X}.W \leq \lambda \vec{X}.W' : \vec{\kappa} \rightarrow \kappa'$.

We have by induction hypothesis $\Gamma, X_1 : \kappa_1 \vdash \lambda X_2 \dots \lambda X_n.W \leq \lambda X_2 \dots \lambda X_n.W' : \kappa_2 \rightarrow \dots \rightarrow \kappa_n \rightarrow \kappa'$ so we can apply (S-ABS):

$$\frac{\Gamma, X_1 : \kappa_1 \vdash \lambda X_2 \dots \lambda X_n.W \leq \lambda X_2 \dots \lambda X_n.W' : \kappa_2 \rightarrow \dots \rightarrow \kappa_n \rightarrow \kappa'}{\Gamma \vdash \lambda \vec{X}.W \leq \lambda \vec{X}.W' : \vec{\kappa} \rightarrow \kappa'} \text{ (S-ABS)} \quad \square$$

Proof. of rule (S-APP-VECT) by induction on $|\vec{G}|$.

Case $|\vec{G}| = 0$. Nothing to show.

Case $|\vec{G}| = n$. We have $\Gamma \vdash F \leq F' : \vec{\kappa} \rightarrow \kappa'$ and $\Gamma \vdash G_i : \kappa_i$ and we show $\Gamma \vdash F\vec{G} \leq F'\vec{G} : \kappa'$.

We have by induction hypothesis $\Gamma \vdash F G_1 \dots G_{n-1} \leq F' G_1 \dots G_{n-1} : \kappa_n \rightarrow \kappa'$ so we can apply (S-APP):

$$\frac{\Gamma \vdash F G_1 \dots G_{n-1} \leq F' G_1 \dots G_{n-1} : \kappa_n \rightarrow \kappa' \quad \Gamma \vdash G_n : \kappa_n}{\Gamma \vdash F\vec{G} \leq F'\vec{G} : \kappa'} \text{ (S-APP)} \quad \square$$

We also need to show that subtyping is compatible with kinding.

Lemma 1.5.1 (Validity of subtyping)

If $\mathcal{D} :: \Gamma \vdash F \leq F' : \kappa$ then $\Gamma \vdash F : \kappa$ and $\Gamma \vdash F' : \kappa$.

Proof.

By induction on \mathcal{D} .

Case (S-VAR) We have $\Gamma \vdash X \leq G : \kappa$ and we show $\Gamma \vdash X : \kappa$ and $\Gamma \vdash G : \kappa$. We have by assumption $\Gamma \vdash$ and $X \leq G : \kappa \in \Gamma$. So we apply (K-VAR-BOUND) and we get $\Gamma \vdash X : \kappa$. Moreover we have by lemma 1.2.3, $\Gamma \vdash G : \kappa$.

Case (S-APP) We have $\Gamma \vdash FA \leq GA : \kappa'$ and we show $\Gamma \vdash FA : \kappa'$ and $\Gamma \vdash GA : \kappa'$. By induction hypothesis we have $\Gamma \vdash F : \kappa \rightarrow \kappa'$ and $\Gamma \vdash G : \kappa \rightarrow \kappa'$ and by assumption we have $\Gamma \vdash A : \kappa$. So we apply in both cases (K-APP).

Case (S-ABS) We have $\Gamma \vdash \lambda XF \leq \lambda XF' : \kappa \rightarrow \kappa'$ and we show $\Gamma \vdash \lambda XF : \kappa \rightarrow \kappa'$ and $\Gamma \vdash \lambda XF' : \kappa \rightarrow \kappa'$. By induction hypothesis we have $\Gamma, X : \kappa \vdash F : \kappa'$ and $\Gamma, X : \kappa \vdash F' : \kappa'$ and we finish in both cases with the rule (K-ABS).

Case (S-ARROW) We have $\Gamma \vdash A \rightarrow B \leq A' \rightarrow B' : *$ and we show $\Gamma \vdash A \rightarrow B : *$ and $\Gamma \vdash A' \rightarrow B' : *$. By induction hypothesis we have $\Gamma \vdash A : *$ and $\Gamma \vdash B : *$ and $\Gamma \vdash A' : *$ and $\Gamma \vdash B' : *$, so we finish in both cases by rule (K-ARROW).

Case (S-ALL) We have $\Gamma \vdash \forall X \leq G : \kappa. A \leq \forall X \leq G' : \kappa. A' : *$ and we show $\Gamma \vdash \forall X \leq G : \kappa. A : *$ and $\Gamma \vdash \forall X \leq G' : \kappa. A' : *$. By induction hypothesis we have $\Gamma, X \leq G : \kappa \vdash A : *$ and $\Gamma, X \leq G' : \kappa \vdash A' : *$. In both cases we finish by (K-ALL).

Case (S-TOP) We have $\Gamma \vdash A \leq \top : *$ and we show $\Gamma \vdash A : *$ and $\Gamma \vdash \top : *$. We have the first already by assumption, and by lemma 1.2.2 part (1) we get $\Gamma \vdash$, so we can apply rule (K-TOP) to get $\Gamma \vdash \top : *$.

Case (S-EQ) We have $\Gamma \vdash F \leq G : \kappa$ and we show $\Gamma \vdash F : \kappa$ and $\Gamma \vdash G : \kappa$. By assumption we have $\Gamma \vdash F = G : \kappa$ so we are done by validity of equality (lemma 1.4.3).

Case (S-TRANS) We have $\Gamma \vdash F \leq H : \kappa$ and we show $\Gamma \vdash F : \kappa$ and $\Gamma \vdash H : \kappa$. We are done with the induction hypothesis. □

Lemma 1.5.2 (Admissible rules for subtyping)

1. *Weakening:* If $\mathcal{D} :: \Gamma, \Gamma' \vdash F \leq F' : \kappa$ then $\Gamma, X \leq G : \kappa', \Gamma' \vdash F \leq F' : \kappa$.
2. *Strengthening:* If $\mathcal{D} :: \Gamma, X \leq G : \kappa, \Gamma' \vdash F \leq F' : \kappa'$ and $X \notin \text{FV}(F, F', \Gamma')$, then $\Gamma \vdash F \leq F' : \kappa'$.

Proof. By induction on \mathcal{D} . □

1.6. Examples

After defining the system $F_{<}^\omega$ formally, we now show some examples of types and subtype relations between types that can be defined with the system as an intuition of its expressive power.

Define the existential quantifier $\exists X : *. A$ as follows:

$$\exists X : *. A = (\lambda F. (\forall Y : *. (\forall X : *. F X \rightarrow Y) \rightarrow Y)) (\lambda X A)$$

Pair can also be defined as follows:

$$\text{Pair} = \lambda T_1 \lambda T_2. (\forall X : *. (T_1 \rightarrow T_2 \rightarrow X) \rightarrow X)$$

Assume a fixed, finite set of labels $\{l_1, \dots, l_n\} =: L$

Write:

$$\{T_i^{i \in 1..n}\} := \text{Pair } T_1 (\text{Pair } T_2 \dots (\text{Pair } T_n \top) \dots)$$

or more formally,

$$\begin{aligned} \{\} &= \top \\ \{t_i : T_i^{i \in 1..n}\} &= \text{Pair } T_1 \{T_i^{i \in 2..n}\} \end{aligned}$$

Record type $\{t_1 : A_1, \dots, t_k : A_k\}$ where $\{t_1, \dots, t_k\} \subset L$ (t_i distinct) can be encoded as $\{T_i^{i=1, \dots, n}\}$ where $T_i = A_j$ if $t_j = l_i$, $T_i = \top$ if $\nexists j. t_j = l_i$. Then the usual subtyping rules for records are admissible, e.g. $\{l_1 : A_1, l_2 : A_2\} \leq \{l_1 : A_1\}$.

With the existential quantifier and with records one can define **Object**:

$$\text{Object} = \lambda M : * \rightarrow *. \exists R : *. \{\text{fields} : R, \text{methods} : MR\}$$

Object is a type operator of kind $(* \rightarrow *) \rightarrow *$ that captures the common structure of all object types. For a more detailed explanation and examples of the use of **Object** see [Pie02, Chapter 32].

Our goal is to derive with the subtyping rules the following rule

$$\frac{M \leq M'}{\text{Object } M \leq \text{Object } M'}$$

First, we define auxiliary rules about pairs, records and existential quantifiers.

Lemma 1.6.1 *The following rule is derivable:*

$$\frac{\Gamma \vdash A \leq B : * \quad \Gamma \vdash C \leq D : *}{\Gamma \vdash \text{Pair } A C \leq \text{Pair } B D : *} \quad (\text{PAIR-SUB})$$

Proof.

$$\frac{\Gamma, X : * \vdash A \leq B : * \quad \frac{\Gamma, X : * \vdash C \leq D : * \quad \Gamma, X : * \vdash X \leq X : *}{\Gamma, X : * \vdash D \rightarrow X \leq C \rightarrow X : *}}{\Gamma, X : * \vdash B \rightarrow D \rightarrow X \leq A \rightarrow C \rightarrow X} \quad \frac{\Gamma, X : * \vdash X \leq X : *}{\Gamma, X : * \vdash X \leq X : *}}{\frac{\Gamma, X : * \vdash (A \rightarrow C \rightarrow X) \rightarrow X \leq (B \rightarrow D \rightarrow X) \rightarrow X : *}{\Gamma \vdash \forall X : *. (A \rightarrow C \rightarrow X) \rightarrow X \leq \forall X : *. (B \rightarrow D \rightarrow X) \rightarrow X : *}}{\Gamma \vdash \text{Pair } A C \leq \text{Pair } B D : *} \quad (\text{S-ALL}) \quad \text{Def.}$$

□

Lemma 1.6.2 *The following rule is derivable:*

$$\frac{\Gamma, X:* \vdash B \leq C:*}{\Gamma \vdash \exists X.B \leq \exists X.C:*} \text{ (EX-SUB)}$$

Proof.

$$\frac{\frac{\frac{\Gamma, Y:*, X:* \vdash B \leq C:*}{\Gamma, Y:*, X:* \vdash C \rightarrow Y \leq B \rightarrow Y:*} \quad \overline{\Gamma, Y:* \vdash Y \leq Y:*}}{\Gamma, Y:* \vdash \forall X:*.C \rightarrow Y \leq \forall X:*.B \rightarrow Y} \quad \overline{\Gamma, Y:* \vdash Y \leq Y:*}}{\Gamma, Y:* \vdash (\forall X:*.B \rightarrow Y) \rightarrow Y \leq (\forall X:*.C \rightarrow Y) \rightarrow Y} \text{ Def.}}{\Gamma \vdash \exists X:*.B \leq \exists X:*.C:*}$$

□

Lemma 1.6.3 *The following rule is derivable:*

$$\frac{\Gamma \vdash S_i \leq T_i^{i \in 1..n}}{\Gamma \vdash \{S_i^{i \in 1..n+k}\} \leq \{T_i^{i \in 1..n}\}} \text{ (REC-SUB)}$$

Proof. By induction on n :

Case $n = 0$ We show:

$$\overline{\Gamma \vdash \{S_i^{i \in 1..k}\} \leq \{\}} \text{ (REC-SUB)}$$

By definition this is equivalent to

$$\overline{\Gamma \vdash \{S_i^{i \in 1..k}\} \leq \top} \text{ (REC-SUB)}$$

and it follows by (S-Top).

Case $n \rightarrow n + 1$ We show

$$\overline{\Gamma \vdash \{S_i^{i \in 1..n+1+k}\} \leq \{T_i^{i \in 1..n+1}\}}$$

We derive:

$$\frac{\frac{\Gamma \vdash S_1 \leq T_1:* \quad \frac{\Gamma \vdash S_i \leq T_i^{i \in 2..n+1}}{\Gamma \vdash \{S_i^{i \in 2..n+1+k}\} \leq \{T_i^{i \in 2..n+1}\}:*} \text{ I.V.}}{\Gamma \vdash \text{Pair } S_1 \{S_i^{i \in 2..n+1+k}\} \leq \text{Pair } T_1 \{T_i^{i \in 2..n+1}\}:*} \text{ (PAIR-SUB)}}{\Gamma \vdash \{S_i^{i \in 1..n+1+k}\} \leq \{T_i^{i \in 1..n+1}\}} \text{ Def.}$$

□

Now we can finally derive the rule (OBJECT-SUB).

Lemma 1.6.4 *The following rule is derivable:*

$$\frac{\Gamma \vdash M \leq M': * \rightarrow *}{\Gamma \vdash \text{Object } M \leq \text{Object } M': *}$$
 (OBJECT-SUB)

Proof.

$$\begin{aligned} \text{Object } M &= \exists R: *. \{\text{fields}: R, \text{methods}: M R\} \\ \text{Object } M' &= \exists R: *. \{\text{fields}: R, \text{methods}: M' R\} \end{aligned}$$

And we derive:

$$\frac{\frac{\frac{\Gamma, R: * \vdash R \leq R: *}{\Gamma, R: * \vdash R \leq R: *}}{\Gamma, R: * \vdash \{\text{fields}: R, \text{methods}: M R\} \leq \{\text{fields}: R, \text{methods}: M' R\}: *}}{\Gamma \vdash \exists R: *. \{\text{fields}: R, \text{methods}: M R\} \leq \exists R: *. \{\text{fields}: R, \text{methods}: M' R\}: *}} \text{(EX-SUB)}$$

$$\frac{\Gamma, R: * \vdash M \leq M': * \rightarrow * \quad \Gamma, R: * \vdash R \leq R: *}{\Gamma, R: * \vdash M R \leq M' R: *}$$
 (REC-SUB)

□

Counter example This is the type of purely functional Counter objects:

$$\text{Counter} = \{\exists X. \{\text{fields}: X, \text{methods}: \{\text{get}: X \rightarrow \text{Nat}, \text{inc}: X \rightarrow X\}\}\}$$

The elements of this type are packages with a hidden state type X , a state of type X , and a record of methods of type $\{\text{get}: X \rightarrow \text{Nat}, \text{inc}: X \rightarrow X\}$.

ResetCounter is a subtype of Counter, i.e., $\text{ResetCounter} \leq \text{Counter}$:

$$\text{ResetCounter} = \{\exists X. \{\text{fields}: X, \text{methods}: \{\text{get}: X \rightarrow \text{Nat}, \text{inc}: X \rightarrow X, \text{reset}: X \rightarrow X\}\}\}$$

Using Object, we can express Counter as the combination of two pieces: $\text{Counter} = \text{Object CounterM}$ where

$$\text{CounterM} = \lambda R. \{\text{get}: R \rightarrow \text{Nat}, \text{inc}: R \rightarrow R\}$$

is a type operator of kind $* \rightarrow *$ representing the specific method interface of counter objects.

The same way we can define ResetCounter:

$$\begin{aligned} \text{ResetCounterM} &= \lambda R. \{\text{get}: R \rightarrow \text{Nat}, \text{inc}: R \rightarrow R, \text{reset}: R \rightarrow R\} \\ \text{ResetCounter} &= \text{Object ResetCounterM} \end{aligned}$$

The next goal is to show $\text{ResetCounter} \leq \text{Counter}$. For it we show first: $\text{ResetCounterM} \leq \text{CounterM}$, and apply (OBJECT-SUB).

$$\frac{\text{ResetCounterM} \leq \text{CounterM}: * \rightarrow *}{\text{Object ResetCounterM} \leq \text{Object CounterM}: *}$$
 (OBJECT-SUB)

Lemma 1.6.5 *The following is provable: $\text{ResetCounterM} \leq \text{CounterM}$.*

Proof.

$$\begin{aligned} \text{CounterM} &= \lambda R. \{\text{get}: R \rightarrow \text{Nat}, \text{inc}: R \rightarrow R\}: * \rightarrow * \\ \text{ResetCounterM} &= \lambda R. \{\text{get}: R \rightarrow \text{Nat}, \text{inc}: R \rightarrow R, \text{reset}: R \rightarrow R\}: * \rightarrow * \end{aligned}$$

We derive:

$$\frac{\frac{\frac{R: * \vdash R \leq R: *}{R: * \vdash R \rightarrow \text{Nat} \leq R \rightarrow \text{Nat}: *} \quad \frac{R: * \vdash \text{Nat} \leq \text{Nat}: *}{R: * \vdash R \rightarrow R \leq R \rightarrow R: *}}{R: * \vdash \{\text{get}: R \rightarrow \text{Nat}, \text{inc}: R \rightarrow R, \text{reset}: R \rightarrow R\} \leq \{\text{get}: R \rightarrow \text{Nat}, \text{inc}: R \rightarrow R\}: *} \quad \frac{R: * \vdash R \leq R: * \quad R: * \vdash R \leq R: *}{R: * \vdash R \rightarrow R \leq R \rightarrow R: *}}{\vdash \lambda R. \{\text{get}: R \rightarrow \text{Nat}, \text{inc}: R \rightarrow R, \text{reset}: R \rightarrow R\} \leq \lambda R. \{\text{get}: R \rightarrow \text{Nat}, \text{inc}: R \rightarrow R\}: * \rightarrow *} \quad \square$$

Now suppose we have functions of the following types:

$$A := \forall X \leq \text{ResetCounterM}: * \rightarrow *. \text{Counter} \rightarrow \text{Object } X$$

$$B := \forall X \leq \text{ResetCounterM}: * \rightarrow *. \text{Object } X \rightarrow \text{Object } X$$

and that we want to derive $A \leq B$. We can derive it in the following way:

(We abbreviate `ResetCounterM` as `RCM`, `CounterM` as `CM` and `Object` as `O` for lack of space in the derivation.)

$$\frac{\frac{\frac{X \leq \text{RCM} \vdash X \leq \text{RCM}: * \rightarrow * \quad \vdash \text{RCM} \leq \text{CM}: * \rightarrow *}{X \leq \text{RCM} \vdash X \leq \text{CM}: * \rightarrow *} \quad \frac{X \leq \text{RCM} \vdash X = X: *}{X \leq \text{RCM} \vdash X \leq X: *}}{X \leq \text{RCM} \vdash O X \leq O \text{CM}: *} \quad \frac{X \leq \text{RCM} \vdash X = X: *}{X \leq \text{RCM} \vdash O X \leq O X: *}}{\frac{X \leq \text{RCM}: * \rightarrow * \vdash O \text{CM} \rightarrow O X \leq O X \rightarrow O X: *}{\vdash \forall X \leq \text{RCM}: * \rightarrow *. O \text{CM} \rightarrow O X \leq \forall X \leq \text{RCM}: * \rightarrow *. O X \rightarrow O X: *}}$$

2. Normalization of constructors

In chapter 1 we have introduced system $F_{<}^{\omega}$ with kinding, equality, and subtyping, and have shown some examples of its use. Our global goal in this first part is the definition of algorithmic subtyping rules for the system. The idea of the algorithm is to normalize the constructors and then to apply the algorithm to constructors in normal form. Thus, we first need to define a normalizer for the system, and this is the goal of this chapter. Since we consider $\beta\eta$ -equality, we need η -long β -normal forms.

We aim at proving all the properties of algorithmic subtyping (soundness, completeness and termination) using syntactical proofs, thus, we also need to define a normalization function syntactically. For this purpose we use hereditary substitution, i.e., a substitution that eliminates β -redexes which are created by the substitution, preserving normal forms this way. We provide a normalization algorithm $\text{nf}(\cdot)$ based on hereditary substitution and we show its soundness, i.e., $\Gamma \vdash F = \text{nf}(F) : \kappa$, a constructor F is equal to its normal form $\text{nf}(F)$ and its completeness: $\Gamma \vdash F = F' : \kappa$ implies $F \equiv \text{nf}(F)$, if two constructors are equal, their normal forms are identical.

In order to show completeness of the normalizer, which, again, is the difficult task, and for proving termination of the subtyping algorithm, we introduce another judgement $\Gamma \vdash F \uparrow \kappa$, read "constructor F has kind κ in context Γ and it is η -long β -normal." This judgement characterizes constructors in normal form. We show that this definition is sound, i.e., that it indeed characterizes normal constructors ($\Gamma \vdash U \uparrow \kappa$ implies $\text{nf}(U) \equiv U$), and complete ($\Gamma \vdash F : \kappa$ implies $\text{nf}(\Gamma) \vdash \text{nf}(F) \uparrow \kappa$), i.e., wellkinded constructors in normal form can always be characterized by \uparrow .

2.1. Equality of contexts

We introduce here another auxiliar relation on contexts. We can extend the definition of equality of constructors in the obvious way to contexts, i.e., a context Γ is equal to a context Γ' , $\vdash \Gamma = \Gamma'$ if Γ and Γ' contain the same variables, and the bounds of the variables are $\beta\eta$ -equal in both contexts. This concept extends the concept of similarity of contexts from last chapter, and its purpose is also a similar one. We have shown that kinding and equality are preserved under similar contexts. In the following we show that also subtyping is preserved under equal contexts. Afterwards we show that a context Γ is equal to its normal form $\text{nf}(\Gamma)$, i.e., the same context, after normalizing all the bounds, and this result is useful for proving soundness of algorithmic subtyping in the next chapter.

Equality of contexts $\vdash \Gamma = \Gamma'$

$$\frac{}{\vdash \diamond = \diamond} \text{ (EQC-EMPTY)} \qquad \frac{\vdash \Gamma = \Gamma' \quad \Gamma \vdash G = G' : \kappa}{\vdash \Gamma, X \leq G : \kappa = \Gamma', X \leq G' : \kappa} \text{ (EQC-BOUND)}$$

One obvious property of the equality of contexts is that two equal contexts are also similar, i.e.,

Lemma 2.1.1 *If $\mathcal{D} :: \vdash \Gamma = \Gamma'$ then $\Gamma \simeq \Gamma'$.*

Proof. By induction on \mathcal{D} .

Case (EQC-EMPTY) Nothing to show.

Case (EQC-BOUND) We have $\vdash \Gamma, X \leq G : \kappa = \Gamma', X \leq G' : \kappa$ and we show $\Gamma, X \leq G : \kappa \simeq \Gamma', X \leq G' : \kappa$. By induction hypothesis we have $\Gamma \simeq \Gamma'$, thus, by definition, also $\Gamma, X \leq G : \kappa \simeq \Gamma', X \leq G' : \kappa$. □

Lemma 2.1.2 (Properties of the equality of contexts)

1. *Reflexivity:* If $\Gamma \vdash$ then $\vdash \Gamma = \Gamma$.
2. *Transitivity:* If $\vdash \Gamma_1 = \Gamma_2$ and $\vdash \Gamma_2 = \Gamma_3$ then $\vdash \Gamma_1 = \Gamma_3$.
3. *Symmetry:* If $\vdash \Gamma = \Gamma'$ then $\vdash \Gamma' = \Gamma$.
4. *Validity:* If $\mathcal{D} :: \vdash \Gamma = \Gamma'$ then $\Gamma \vdash$ and $\Gamma' \vdash$.

Proof. The properties (1) to (3) are inherited from the equality of constructors. The validity property (4) is proven by induction on \mathcal{D} .

Case (EQC-EMPTY) We have $\vdash \diamond = \diamond$ and we show $\diamond \vdash$, and it follows by (C-EMPTY).

Case (EQC-BOUND) We have $\vdash \Gamma, X \leq G : \kappa = \Gamma', X \leq G' : \kappa$ and we show $\Gamma, X \leq G : \kappa \vdash$ and $\Gamma', X \leq G' : \kappa \vdash$. By induction hypothesis we have $\Gamma \vdash$ and $\Gamma' \vdash$. And we finish with (C-BOUND). □

Lemma 2.1.3 *If $(X \leq G : \kappa) \in \Gamma$ and $\mathcal{D} :: \vdash \Gamma = \Gamma'$ then there is a G' with $X \leq G' : \kappa \in \Gamma'$ and $\Gamma \vdash G = G' : \kappa$.*

Proof. By induction on \mathcal{D} .

Case (EQC-EMPTY) Impossible.

Case (EQC-BOUND) We have $\vdash \Gamma, Y \leq H : \kappa_1 = \Gamma', Y \leq H' : \kappa_1$ and $(X \leq G : \kappa) \in \Gamma, Y \leq H' : \kappa_1$ and we show: there is a G' with $X \leq G' : \kappa \in \Gamma', Y \leq H' : \kappa_1$ and $\Gamma, Y \leq H' : \kappa_1 \vdash G = G' : \kappa$. If $X = Y$ then $G = H$ and $G' = H'$. Otherwise we have by induction hypothesis a G' with $X \leq G' : \kappa \in \Gamma'$ and $\Gamma \vdash G = G' : \kappa$, and this is our witness, since $X \leq G' : \kappa \in \Gamma', Y \leq H : \kappa_1$ as well, and $\Gamma, Y \leq H : \kappa_1 \vdash G = G' : \kappa$ by weakening.

□

We have already seen that kinding and equality are preserved under similar contexts. The notion of equality of contexts also preserves subtyping. Similar contexts are not enough for preserving subtyping, because for deciding subtyping we do look at the bounds, so just having some bound is not enough, but it is enough to have equal bounds.

Lemma 2.1.4 (Preservation of judgements under equality of contexts)

1. If $\mathcal{D} :: \Gamma \vdash F : \kappa$ and $\vdash \Gamma = \Gamma'$ then $\Gamma' \vdash F : \kappa$.
2. If $\mathcal{D} :: \Gamma \vdash F = F' : \kappa$ and $\vdash \Gamma = \Gamma'$ then $\Gamma' \vdash F = F' : \kappa$.
3. If $\mathcal{D} :: \Gamma \vdash F \leq F' : \kappa$ and $\vdash \Gamma = \Gamma'$ then $\Gamma' \vdash F \leq F' : \kappa$.

Proof.

1. We have by lemma 2.1.1 $\Gamma \simeq \Gamma'$ and by lemma 1.3.1 follows $\Gamma' \vdash F : \kappa$.
2. We have by lemma 2.1.1 $\Gamma \simeq \Gamma'$ and by lemma 1.4.2 follows $\Gamma' \vdash F = F' : \kappa$.
3. By induction on \mathcal{D} . The interesting case is:

Case (S-VAR). We have $\Gamma \vdash X \leq G : \kappa$ and $\vdash \Gamma = \Gamma'$ and we show $\Gamma' \vdash X \leq G : \kappa$. By lemma 2.1.3 there is a G' with $X \leq G' \in \Gamma'$ and $\Gamma \vdash G = G' : \kappa$ and we have $\Gamma' \vdash$ by lemma 2.1.2, part 4. So we can derive:

$$\frac{\Gamma' \vdash \frac{X \leq G' : \kappa \in \Gamma'}{\Gamma' \vdash X \leq G' : \kappa} \text{ (S-VAR)} \quad \frac{\frac{\frac{\Gamma \vdash G = G' : \kappa}{\Gamma' \vdash G = G' : \kappa} \text{ part (2)}}{\Gamma' \vdash G' = G : \kappa} \text{ (EQ-SYM)}}{\Gamma' \vdash G' \leq G : \kappa} \text{ (S-EQ)}}{\Gamma' \vdash X \leq G : \kappa} \text{ (EQ-TRANS)}$$

□

2.2. Hereditary substitution

We define a 4-ary function $[G^\kappa/X]F$, called hereditary substitution, which returns a result \hat{F} . A result is either just a constructor F or a constructor annotated with a kind, written F^κ . The intention is that if G and F are β -normal (and well typed) then the result will also be β -normal (and well typed). Our definition is an extension of Abel's hereditary substitution for the simply typed λ -calculus [Abe06a]. This kind of substitution was introduced by Watkins et al. in [WCPW03].

Sometimes we discard the type annotation of the results. To formalize this we define the operation \underline{F}^κ by $\underline{F}^\kappa = F$ and $\underline{F} = F$. This operation is to be applied implicitly when the context demands it. For example, application of two results implicitly discards the type annotations: $\hat{F}\hat{H} = \underline{\hat{F}}\underline{\hat{H}}$. Finally, reannotation $\hat{F}^\kappa = (\underline{\hat{F}})^\kappa$ puts a fresh type annotation κ onto a result.

Hereditary substitution $\boxed{[G^\kappa/X]F}$

$$\begin{aligned}
[G^\kappa/X]Y &:= G^\kappa && \text{if } X = Y \\
&Y && \text{otherwise} \\
[G^\kappa/X](\lambda Y F) &:= \lambda Y.[G^\kappa/X]F && \text{where } Y \text{ fresh for } X, G. \\
[G^\kappa/X](FH) &:= \begin{array}{l} ([\hat{H}^{\kappa_1}/Y]F')^{\kappa_2} \\ \hat{F} \hat{H} \end{array} && \begin{array}{l} \text{if } \hat{F} = (\lambda Y F')^{\kappa_1 \rightarrow \kappa_2} \\ \text{otherwise, where } \hat{F} = [G^\kappa/X]F \\ \hat{H} = [G^\kappa/X]H \end{array} \\
[G^\kappa/X](A \rightarrow B) &:= [G^\kappa/X]A \rightarrow [G^\kappa/X]B \\
[G^\kappa/X](\forall Y \leq H : \kappa. A) &:= \forall Y \leq [G^\kappa/X]H : \kappa. [G^\kappa/X]A : * && \text{where } Y \text{ fresh for } X, G \\
[G^\kappa/X]\top &:= \top
\end{aligned}$$

Using hereditary substitution we define the following function, which is used in the definition of normal forms. It can be seen as an application which eliminates redexes by need, i.e., an application compatible with normalization.

$\boxed{F @^\kappa G}$

$$\begin{aligned}
\lambda X F @^\kappa G &:= [G^\kappa/X]F \\
N @^\kappa G &:= NG
\end{aligned}$$

$\boxed{F @^{\vec{\kappa}} \vec{G}}$

$$F @^{\vec{\kappa}} \vec{G} := (((F @^{\kappa_1} G_1) @^{\kappa_2} G_2) \dots @^{\kappa_n} G_n)$$

Notice that with this notation we can write $[G^\kappa/X](FH)$ more compactly:

$$[G^\kappa/X](FH) = [G^\kappa/X]F @^{\kappa_1} [G^\kappa/X]H$$

Hereditary substitution can be extended to contexts in the same way as the usual substitution, i.e., substitution in every bound in context.

Hereditary substitution for contexts $\boxed{[G^\kappa/X]\Gamma}$

$$\begin{aligned}
[G^\kappa/X]\diamond &= \diamond \\
[G^\kappa/X](\Gamma, Y \leq H : \kappa) &= [G^\kappa/X]\Gamma, Y \leq [G^\kappa/X]H
\end{aligned}$$

Now the goal is to show that hereditary substitution is the same operation as conventional substitution modulo equality.

Lemma 2.2.1 (Invariant) *If $[G^\kappa/X]F = F^{\kappa_2}$ then $|\kappa_2| \leq |\kappa|$.*

Proof. By induction on F .

There are only two cases which return an annotated term:

Case $[G^\kappa/X]X = G^\kappa$. Trivially, since $|\kappa| \leq |\kappa|$.

Case $[G^\kappa/X](FH) = ([\hat{H}^{\kappa_1}/Y]F')^{\kappa_2}$ where $[G^\kappa/X]F = (\lambda Y.F')^{\kappa_1 \rightarrow \kappa_2}$. By induction hypothesis $|\kappa_1 \rightarrow \kappa_2| \leq |\kappa|$. This proves the invariant, since $|\kappa_2| \leq |\kappa_1 \rightarrow \kappa_2|$ by definition of the measure $|\cdot|$. □

Lemma 2.2.2 (Termination and soundness) *If $\Gamma, X : \kappa, \Gamma' \vdash F : \kappa'$ and $\Gamma \vdash G : \kappa$ then $\Gamma, [G/X]\Gamma' \vdash [G^\kappa/X]F = [G/X]F : \kappa'$.*

Proof. By lexicographical induction on $(|\kappa|, F)$. Let $\Gamma, [G/X]\Gamma' = \hat{\Gamma}$

Case (K-VAR) We have $\Gamma, X : \kappa, \Gamma' \vdash Y : \kappa'$ and $[G^\kappa/X]Y$. Termination is obvious. For soundness we show $\hat{\Gamma} \vdash [G^\kappa/X]Y = [G/X]Y : \kappa'$.

Subcase $X = Y$. Then $[G^\kappa/X]Y = G = [G/X]Y$, so by rule (EQ-REFL) $\Gamma \vdash G = G : \kappa'$ and by weakening $\hat{\Gamma} \vdash G = G' : \kappa$.

Subcase $X \neq Y$. Then $[G^\kappa/X]Y = Y = [G/X]Y$, so by rule (EQ-REFL) $\hat{\Gamma} \vdash Y = Y : \kappa'$.

Case (K-ABS) We have $\Gamma, X : \kappa, \Gamma' \vdash \lambda Y F : \kappa_1 \rightarrow \kappa_2$ and $[G^\kappa/X](\lambda Y F)$. By induction hypothesis holds that $[G^\kappa/X]F$ terminates, so $[G^\kappa/X](\lambda Y F)$ terminates as well. For soundness, by induction hypothesis holds $\hat{\Gamma}, Y : \kappa_1 \vdash [G^\kappa/X]F = [G/X]F : \kappa_2$.

$$\frac{\frac{\hat{\Gamma}, Y : \kappa_1 \vdash [G^\kappa/X]F = [G/X]F : \kappa_2}{\hat{\Gamma} \vdash \lambda Y.[G^\kappa/X]F = \lambda Y.[G/X]F : \kappa_2} \text{ (EQ-}\lambda\text{)}}{\hat{\Gamma} \vdash [G^\kappa/X](\lambda Y F) = [G/X](\lambda Y F) : \kappa_2} \text{ By definition}$$

Case (K-ARROW) We have $\Gamma, X : \kappa, \Gamma' \vdash A \rightarrow B : *$ and $[G^\kappa/X](A \rightarrow B)$. By induction hypothesis holds that $[G^\kappa/X]A$ and $[G^\kappa/X]B$ terminate, so $[G^\kappa/X](A \rightarrow B)$ terminates as well. In order to show soundness we notice that by induction hypothesis holds $\hat{\Gamma} \vdash [G^\kappa/X]A = [G/X]A : *$ and $\hat{\Gamma} \vdash [G^\kappa/X]B = [G/X]B : *$.

$$\frac{\frac{\hat{\Gamma} \vdash [G^\kappa/X]A = [G/X]A : * \quad \hat{\Gamma} \vdash [G^\kappa/X]B = [G/X]B : *}{\hat{\Gamma} \vdash [G^\kappa/X]A \rightarrow [G^\kappa/X]B = [G/X]A \rightarrow [G/X]B : *} \text{ (EQ-ARROW)}}{\hat{\Gamma} \vdash [G^\kappa/X](A \rightarrow B) = [G/X](A \rightarrow B) : *} \text{ By definition}$$

Case (K-ALL) We have $\Gamma, X : \kappa, \Gamma' \vdash \forall Y \leq H : \kappa'. A : *$ and $[G^\kappa/X](\forall Y \leq H : \kappa'. A)$. By induction hypothesis holds that $[G^\kappa/X]H$ and $[G^\kappa/X]A$ terminate, so $[G^\kappa/X](\forall Y \leq H : \kappa'. A)$ terminates as well. For soundness we look at the induction hypothesis: $\hat{\Gamma} \vdash [G^\kappa/X]H = [G/X]H : \kappa'$ and $\hat{\Gamma}, Y \leq [G/X]H : \kappa' \vdash [G^\kappa/X]A = [G/X]A : *$ thus, we can derive:

$$\begin{array}{c}
\frac{\hat{\Gamma} \vdash [G^\kappa/X]H = [G/X]H:\kappa}{\hat{\Gamma} \vdash [G/X]H = [G^\kappa/X]H:\kappa} \quad \frac{\hat{\Gamma}, Y \leq [G/X]H:\kappa' \vdash [G^\kappa/X]A = [G/X]A:*\ }{\hat{\Gamma}, Y \leq [G/X]H:\kappa' \vdash [G/X]A = [G^\kappa/X]A:*\ } \\
\hline
\frac{\hat{\Gamma} \vdash \forall Y \leq [G/X]H:\kappa'. [G/X]A = \forall Y \leq [G^\kappa/X]H:\kappa'. [G^\kappa/X]A:*\ }{\hat{\Gamma} \vdash \forall Y \leq [G^\kappa/X]H:\kappa'. [G^\kappa/X]A = \forall Y \leq [G/X]H:\kappa'. [G/X]A:*\ } \text{ (EQ-SYM)} \\
\hline
\hat{\Gamma} \vdash [G^\kappa/X](\forall Y \leq H:\kappa'. A) = [G/X](\forall Y \leq H:\kappa'. A):*\ \text{ By definition}
\end{array}$$

Case (K-TOP) We have $\Gamma, X:\kappa, \Gamma' \vdash \top:*$ and $[G^\kappa/X]\top$. Termination is obvious and, since $[G^\kappa/X]\top = \top = [G/X]\top$ by definition, we have by (EQ-REFL) $\hat{\Gamma} \vdash \top = \top:*$.

Case (K-APP) We have $\Gamma, X:\kappa, \Gamma' \vdash FH:\kappa_2$ and $[G^\kappa/X](FH)$. By induction hypothesis, $\hat{F} = [G^\kappa/X]F$ and $\hat{H} = [G^\kappa/X]H$ are both defined.

Subcase $\hat{F} \neq (\lambda Y F')^{\kappa_1 \rightarrow \kappa_2}$ Then $[G^\kappa/X](FH)$ terminates. Moreover, for soundness, by induction hypothesis holds $\hat{\Gamma} \vdash [G^\kappa/X]F = [G/X]F:\kappa_1 \rightarrow \kappa_2$ and $\hat{\Gamma} \vdash [G^\kappa/X]H = [G/X]H:\kappa_1$.

$$\begin{array}{c}
\frac{\hat{\Gamma} \vdash [G^\kappa/X]F = [G/X]F:\kappa_1 \rightarrow \kappa_2 \quad \hat{\Gamma} \vdash [G^\kappa/X]H = [G/X]H:\kappa_1}{\hat{\Gamma} \vdash ([G^\kappa/X]F)([G^\kappa/X]H) = ([G/X]F)([G/X]H):\kappa_2} \text{ (EQ-APP)} \\
\hline
\hat{\Gamma} \vdash [G^\kappa/X](FH) = [G/X](FH):\kappa_2 \text{ By definition}
\end{array}$$

Subcase $\hat{F} = (\lambda Y F')^{\kappa_1 \rightarrow \kappa_2}$

Using the invariant, we infer $|\kappa_1| < |\kappa_1 \rightarrow \kappa_2| \leq |\kappa|$. Hence, we can again apply the induction hypothesis to infer that $[\hat{H}^{\kappa_1}/Y]F'$ terminates, thus, by definition, also $[G^\kappa/X](FH)$. For soundness we have the following induction hypothesis:

1. $\hat{\Gamma} \vdash \lambda Y F' = [G/X]F:\kappa_1 \rightarrow \kappa_2$
2. $\hat{\Gamma} \vdash \hat{H} = [G/X]H:\kappa_1$
3. $\hat{\Gamma} \vdash [\hat{H}^{\kappa_1}/Y]F' = [\hat{H}/Y]F':\kappa_2$

With them we can derive:

$$\begin{array}{c}
\frac{\hat{\Gamma} \vdash \lambda Y F' = [G/X]F:\kappa_1 \rightarrow \kappa_2 \quad \hat{\Gamma} \vdash \hat{H} = [G/X]H:\kappa_1}{\hat{\Gamma} \vdash (\lambda Y F')\hat{H} = ([G/X]F)([G/X]H):\kappa_2} \text{ (EQ-APP)} \\
\hline
\text{(I)} \hat{\Gamma} \vdash ([G/X]F)([G/X]H) = (\lambda Y F')\hat{H}:\kappa_2 \text{ (EQ-SYM)} \\
\hline
\frac{\hat{\Gamma}, Y:\kappa \vdash F':\kappa_2 \quad \hat{\Gamma} \vdash \hat{H}:\kappa_1}{\hat{\Gamma} \vdash (\lambda Y F')\hat{H} = [\hat{H}/Y]F':\kappa_2} \\
\hline
\text{(II)} \hat{\Gamma} \vdash ([G/X]F)([G/X]H) = [\hat{H}/Y]F':\kappa_2 \\
\hline
\frac{\hat{\Gamma} \vdash [\hat{H}^{\kappa_1}/Y]F' = [\hat{H}/Y]F':\kappa_2}{\hat{\Gamma} \vdash ([G/X]F)([G/X]H) = [\hat{H}^{\kappa_1}/Y]F':\kappa_2} \text{ By definition} \\
\hline
\frac{\hat{\Gamma} \vdash [G/X](FH) = [G^\kappa/X](FH):\kappa_2}{\hat{\Gamma} \vdash [G^\kappa/X](FH) = [G/X](FH):\kappa_2} \text{ (EQ-SYM)}
\end{array}$$

□

Corollary 2.2.3 *If $\Gamma \vdash F : \kappa \rightarrow \kappa'$ and $\Gamma \vdash G : \kappa$ then $\Gamma \vdash F @^\kappa G = F G : \kappa'$.*

Proof.

Case $F = \lambda X F'$. Then we show $\Gamma \vdash \lambda X F' @^\kappa G = (\lambda X F') G : \kappa'$ i.e., $\Gamma \vdash [G^\kappa / X] F' = [G / X] F' : \kappa'$ and this follows by lemma 2.2.2.

Case $F \neq \lambda X F'$. Then we show $\Gamma \vdash F @^\kappa G = F G : \kappa'$ i.e., $\Gamma \vdash F G = F G : \kappa'$ which follows by reflexivity. □

And now we prove another property of hereditary substitution which holds for conventional substitution. This lemma is provably the most difficult one in the whole theory, and the biggest difficulty is to find the right induction measure. It has been shown that $|\kappa| + |\kappa'|$ works. In item 2 we also need a local induction on β -normal constructors. Notice that we require only β -normal constructors for this lemma, not the more specific η -long β -normal constructors.

Lemma 2.2.4 *Let $\Gamma \vdash U : \kappa$ and $\Gamma, X : \kappa, \Gamma' \vdash V : \kappa'$ and $\kappa' = \vec{\kappa} \rightarrow \kappa_0$.*

1. *If $\Gamma, X : \kappa, \Gamma' \vdash W_i : \kappa_i$ for $1 \leq i \leq |\vec{\kappa}|$ then*

$$[U^\kappa / X](V @^{\vec{\kappa}} \vec{W}) \equiv [U^\kappa / X]V @^{\vec{\kappa}} [U^\kappa / X]\vec{W}$$

2. *If $\Gamma, X : \kappa, \Gamma', Y : \kappa', \Gamma'' \vdash W : \kappa''$ then:*

$$[U^\kappa / X]([V^{\kappa'} / Y]W) \equiv [[U^\kappa / X]V^{\kappa'} / Y]([U^\kappa / X]W)$$

Proof. Simultaneously by induction on $|\kappa| + |\kappa'|$, first 1 and then 2.

1. *Case $V \neq \lambda Z V'$.* Then we show

$$\begin{aligned} [U^\kappa / X]V @^{\vec{\kappa}} \vec{W} &\equiv [U^\kappa / X]V @^{\vec{\kappa}} [U^\kappa / X]\vec{W} \\ [U^\kappa / X](V \vec{W}) &\equiv [U^\kappa / X]V @^{\vec{\kappa}} [U^\kappa / X]\vec{W} \end{aligned}$$

and this holds by definition.

Case We show $V = \lambda Z V'$. Then we show

$$[U^\kappa / X](\lambda Z V' @^{\vec{\kappa}} \vec{W}) \equiv [U^\kappa / X](\lambda Z V') @^{\vec{\kappa}} [U^\kappa / X]\vec{W}$$

We have

$$[U^\kappa / X](\lambda Z V' @^{\vec{\kappa}} \vec{W}) \equiv [U^\kappa / X](((W_1^{\kappa_1} / Z)V') @^{\kappa_2, \dots, \kappa_n} W_2, \dots, W_n)$$

and, by IH (1), since $|\kappa| + |\kappa_2, \dots, \kappa_n \rightarrow \kappa_0| < |\kappa| + |\kappa'|$

$$\equiv [U^\kappa / X]([W_1^{\kappa_1} / Z]V') @^{\kappa_2, \dots, \kappa_n} [U^\kappa / X](W_2, \dots, W_n)$$

and, by IH (2), since $|\kappa| + |\kappa_1| < |\kappa| + |\kappa'|$

$$\equiv [[U^\kappa / X]W_1^{\kappa_1} / Z]([U^\kappa / X]V') @^{\kappa_2, \dots, \kappa_n} [U^\kappa / X](W_2, \dots, W_n)$$

and, by definition

$$\equiv (\lambda Z. [U^\kappa / X]V') @^{\vec{\kappa}} [U^\kappa / X]\vec{W}$$

and, by definition

$$\equiv [U^\kappa / X](\lambda Z V') @^{\vec{\kappa}} [U^\kappa / X]\vec{W}$$

and we are done.

2. By local induction on W .

Case $W = Z\vec{W}$, $Z \neq X, Y$. We show

$$\begin{aligned} [U^\kappa/X]([V^{\kappa'}/Y](Z\vec{W})) &\equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X](Z\vec{W})) \\ Z [U^\kappa/X]([V^{\kappa'}/Y]\vec{W}) &\equiv Z [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]\vec{W}) \end{aligned}$$

We have by induction hypothesis

$$[U^\kappa/X]([V^{\kappa'}/Y]W_i) \equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]W_i)$$

for all i , thus, we are finished.

Case $W = Y\vec{W}$ We show

$$\begin{aligned} [U^\kappa/X]([V^{\kappa'}/Y](Y\vec{W})) &\equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X](Y\vec{W})) \text{ i.e.,} \\ [U^\kappa/X](V @^{\vec{\kappa}} [V^{\kappa'}/Y]\vec{W}) &\equiv [[U^\kappa/X]V^{\kappa'}/Y](Y[U^\kappa/X]\vec{W}) \text{ i.e.,} \\ [U^\kappa/X](V @^{\vec{\kappa}} [V^{\kappa'}/Y]\vec{W}) &\equiv [U^\kappa/X]V @^{\vec{\kappa}} [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]\vec{W}) \\ \text{i.e., by IH (2), since } |\kappa| + |\kappa'| &= |\kappa| + |\kappa'| \text{ and } W_i < Y\vec{W} \text{ for all } i \\ [U^\kappa/X](V @^{\vec{\kappa}} [V^{\kappa'}/Y]\vec{W}) &\equiv [U^\kappa/X]V @^{\vec{\kappa}} [U^\kappa/X]([V^{\kappa'}/Y]\vec{W}) \end{aligned}$$

and it holds by induction hypothesis (1) since $|\kappa| + |\kappa'| = |\kappa| + |\kappa'|$.

Case $W = X\vec{W}$ We show

$$\begin{aligned} [U^\kappa/X]([V^{\kappa'}/Y](X\vec{W})) &\equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X](X\vec{W})) \text{ i.e.,} \\ [U^\kappa/X](X[V^{\kappa'}/Y]\vec{W}) &\equiv [[U^\kappa/X]V^{\kappa'}/Y](U @^{\vec{\kappa}} ([U^\kappa/X]\vec{W})) \text{ i.e.,} \\ U @^{\vec{\kappa}} ([U^\kappa/X]([V^{\kappa'}/Y]\vec{W})) &\equiv [[U^\kappa/X]V^{\kappa'}/Y](U @^{\vec{\kappa}} ([U^\kappa/X]\vec{W})) \\ \text{i.e., by IH (1), since } |\kappa'| + |\kappa| &= |\kappa| + |\kappa'| \\ U @^{\vec{\kappa}} ([U^\kappa/X]([V^{\kappa'}/Y]\vec{W})) &\equiv [[U^\kappa/X]V^{\kappa'}/Y]U @^{\vec{\kappa}} [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]\vec{W}) \\ \text{i.e., since } Y \notin \text{FV}(U) & \\ U @^{\vec{\kappa}} ([U^\kappa/X]([V^{\kappa'}/Y]\vec{W})) &\equiv U @^{\vec{\kappa}} [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]\vec{W}) \\ \text{i.e., by IH (2), since } |\kappa| + |\kappa'| &= |\kappa| + |\kappa'| \text{ and } W_i < X\vec{W} \text{ for all } i \\ U @^{\vec{\kappa}} ([U^\kappa/X]([V^{\kappa'}/Y]\vec{W})) &\equiv U @^{\vec{\kappa}} [U^\kappa/X]([V^{\kappa'}/Y]\vec{W}) \end{aligned}$$

and this is trivially true.

Case $W = \top$ We show

$$[U^\kappa/X]([V^{\kappa'}/Y]\top) \equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]\top) \text{ i.e., } \top \equiv \top$$

which is trivially true.

Case $W \equiv W \rightarrow W'$ We show

$$\begin{aligned} [U^\kappa/X]([V^{\kappa'}/Y](W \rightarrow W')) &\equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X](W \rightarrow W')) \\ [U^\kappa/X]([V^{\kappa'}/Y]W) \rightarrow [U^\kappa/X]([V^{\kappa'}/Y]W') &\equiv \\ [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]W) \rightarrow [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]W') & \end{aligned}$$

By induction hypothesis (2) holds

$$\begin{aligned} [U^\kappa/X]([V^{\kappa'}/Y]W) &\equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]W) \text{ and} \\ [U^\kappa/X]([V^{\kappa'}/Y]W') &\equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]W') \end{aligned}$$

thus, we are finished.

Case $W = \lambda ZW$ We show

$$\begin{aligned} [U^\kappa/X]([V^{\kappa'}/Y](\lambda ZW)) &\equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X](\lambda ZW)) \\ \lambda Z.[U^\kappa/X]([V^{\kappa'}/Y]W) &\equiv \lambda Z.[[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]W) \end{aligned}$$

By induction hypothesis (2)

$$[U^\kappa/X]([V^{\kappa'}/Y]W) \equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]W)$$

thus, we are finished.

Case $\forall X \leq U' : \kappa''. V'$ We show

$$\begin{aligned} [U^\kappa/X]([V^{\kappa'}/Y](\forall X \leq U' : \kappa''. V')) &\equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X](\forall X \leq U' : \kappa''. V')) \\ \forall X \leq [U^\kappa/X]([V^{\kappa'}/Y]U') : \kappa''. [U^\kappa/X]([V^{\kappa'}/Y]V') &\equiv \\ \forall X \leq [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]U') : \kappa''. [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]V') &\equiv \end{aligned}$$

By induction hypothesis (2) we have

$$[U^\kappa/X]([V^{\kappa'}/Y]U') \equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]U') \text{ and}$$

$$[U^\kappa/X]([V^{\kappa'}/Y]V') \equiv [[U^\kappa/X]V^{\kappa'}/Y]([U^\kappa/X]V')$$

thus, we are finished. □

2.3. Long normal forms

Wellkinded constructors are β -normalizing, and we can compute their η -long β -normal form. Let $\text{nf}_{\Gamma \vdash \kappa}(F)$ denote the long normal form of F . It is defined when $\Gamma \vdash F : \kappa$. We omit the subscript $\Gamma \vdash \kappa$ if clear from the context of discourse.

The algorithm η -expands the variables. This way it can be guaranteed that every η -long β -normal constructor of kind $\vec{\kappa} \rightarrow *$ with a nonempty vector $\vec{\kappa}$ is an abstraction of the form $\lambda \vec{X}.U$. Moreover, we use hereditary substitution or more exactly the function $@$ for eliminating redexes in the applications.

constructors in normal form $\text{nf}_{\Gamma \vdash \kappa}(F)$

$$\begin{aligned} \text{nf}_{\Gamma \vdash \vec{\kappa} \rightarrow *}(X) &:= \lambda Y_1 \dots \lambda Y_n. X \text{nf}_{\Gamma, \vec{Y} : \vec{\kappa} \vdash \kappa_1}(Y_1) \dots \text{nf}_{\Gamma, \vec{Y} : \vec{\kappa} \vdash \kappa_n}(Y_n) \\ \text{nf}_{\Gamma \vdash *}(T) &:= T \\ \text{nf}_{\Gamma \vdash \kappa \rightarrow \kappa'}(\lambda X F) &:= \lambda X. \text{nf}_{\Gamma, X : \kappa \vdash \kappa'}(F) \\ \text{nf}_{\Gamma \vdash *}(A \rightarrow B) &:= \text{nf}_{\Gamma \vdash *}(A) \rightarrow \text{nf}_{\Gamma \vdash *}(B) \\ \text{nf}_{\Gamma \vdash *}(\forall X \leq G : \kappa. A) &:= \forall X \leq \text{nf}_{\Gamma \vdash \kappa}(G) : \kappa. \text{nf}_{\Gamma, X \leq G : \kappa \vdash *}(A) \\ \text{nf}_{\Gamma \vdash \kappa'}(FG) &:= \text{nf}_{\Gamma \vdash \kappa \rightarrow \kappa'}(F) @^\kappa \text{nf}_{\Gamma \vdash \kappa}(G) \end{aligned}$$

Normalization can be extended to contexts in the obvious way, we write $\text{nf}(\Gamma)$ for the context Γ where all bounds have been normalized.

contexts in normal form $\text{nf}(\Gamma)$

$$\begin{aligned} \text{nf}(\diamond) &= \diamond \\ \text{nf}(\Gamma, X \leq G : \kappa) &= \text{nf}(\Gamma), X \leq \text{nf}_{\Gamma \vdash \kappa}(G) : \kappa \end{aligned}$$

Lemma 2.3.1 *Let $\Gamma \vdash X : \vec{\kappa} \rightarrow *$. Then $\Gamma \vdash X = \text{nf}(X) : \vec{\kappa} \rightarrow *$.*

Proof. By induction on $\vec{\kappa} \rightarrow *$. We have $\Gamma \vdash X : \vec{\kappa} \rightarrow *$ and we show $\Gamma \vdash X = \text{nf}(X) : \vec{\kappa} \rightarrow *$, i.e., $\Gamma \vdash X = \lambda Y_1 \dots \lambda Y_n. X \text{nf}_{\Gamma, \vec{Y} \vdash \vec{\kappa}_1}(Y_1) \dots \text{nf}_{\Gamma, \vec{Y} \vdash \vec{\kappa}_n}(Y_n) : \vec{\kappa} \rightarrow *$.

By induction hypothesis we have $\Gamma, \vec{Y} : \vec{\kappa} \vdash \text{nf}(Y_i) : \kappa_i$ and $\Gamma, \vec{Y} : \vec{\kappa} \vdash X = X : \vec{\kappa} \rightarrow *$ by (EQ-REFL) so we can derive:

$$\frac{\frac{\Gamma \vdash X : \vec{\kappa} \rightarrow *}{\Gamma \vdash X = \lambda \vec{Y}. X \vec{Y} : \vec{\kappa} \rightarrow *}}{\Gamma \vdash X = \lambda \vec{Y}. X \text{nf}(\vec{Y}) : \vec{\kappa} \rightarrow *}}{\frac{\frac{\Gamma, \vec{Y} : \vec{\kappa} \vdash X = X : \vec{\kappa} \rightarrow * \quad \Gamma, \vec{Y} : \vec{\kappa} \vdash Y_i = \text{nf}(Y_i) : \kappa_i}{\Gamma, \vec{Y} : \vec{\kappa} \vdash X \vec{Y} = X \text{nf}(\vec{Y}) : *}}{\Gamma \vdash \lambda \vec{Y}. X \vec{Y} = \lambda \vec{Y}. X \text{nf}(\vec{Y}) : \vec{\kappa} \rightarrow *}}{\Gamma \vdash X = \lambda \vec{Y}. X \text{nf}(\vec{Y}) : \vec{\kappa} \rightarrow *}} \quad \square$$

Lemma 2.3.2 (Soundness and termination of normalization) *Let $\mathcal{D} :: \Gamma \vdash F : \kappa$. Then $\Gamma \vdash F = \text{nf}_{\Gamma \vdash \kappa}(F) : \kappa$.*

Proof. By induction on \mathcal{D} .

Case (K-VAR). It follows by lemma 2.3.1.

Case (K-TOP) We have $\Gamma \vdash \top : *$ and we show $\Gamma \vdash \top = \text{nf}(\top) : *$, i.e., $\Gamma \vdash \top = \top : *$ and it follows by (EQ-REFL).

Case (K-ABS). We have $\Gamma \vdash \lambda X. F : \kappa \rightarrow \kappa'$ and we show $\Gamma \vdash \lambda X. F = \text{nf}(\lambda X. F) : \kappa \rightarrow \kappa'$, i. e. $\Gamma \vdash \lambda X. F = \lambda X. \text{nf}_{\Gamma, X \vdash \kappa'}(F) : \kappa \rightarrow \kappa'$. We have by induction hypothesis $\Gamma, X : \kappa \vdash F = \text{nf}_{\Gamma, X \vdash \kappa'}(F) : \kappa'$ so we apply rule (EQ- λ) and we are done.

Case (K-ARROW) We have $\Gamma \vdash A \rightarrow B : *$ and we show $\Gamma \vdash A \rightarrow B = \text{nf}_{\Gamma \vdash *}(A \rightarrow B) : *$ i.e., $\Gamma \vdash A \rightarrow B = \text{nf}(A) \rightarrow \text{nf}(B) : *$ By induction hypothesis we have $\Gamma \vdash A = \text{nf}(A) : *$ and $\Gamma \vdash B = \text{nf}(B) : *$ and we finish by rule (EQ-ARROW).

Case (K-ALL). We have $\Gamma \vdash \forall X \leq G : \kappa. A : *$ and we show

$$\begin{aligned} \Gamma \vdash \forall X \leq G : \kappa. A &= \text{nf}(\forall X \leq G : \kappa. A) : * \text{ i.e.,} \\ \Gamma \vdash \forall X \leq G : \kappa. A &= \forall X \leq \text{nf}_{\Gamma \vdash \kappa}(G) : \kappa. \text{nf}_{\Gamma, X \leq G \vdash *}(A) : * \end{aligned}$$

We have by induction hypothesis $\Gamma, X \leq G : \kappa \vdash A = \text{nf}_{\Gamma, X \leq G \vdash *}(A) : *$ and $\Gamma \vdash G = \text{nf}_{\Gamma \vdash \kappa}(G) : \kappa$. We finish with the rule (EQ- \forall).

Case (K-APP) We have $\Gamma \vdash FG : \kappa'$. We show $\Gamma \vdash FG = \text{nf}(FG) : \kappa'$. By induction hypothesis we have $\Gamma \vdash F = \text{nf}_{\Gamma \vdash \kappa \rightarrow \kappa'}(F) : \kappa \rightarrow \kappa'$ and $\Gamma \vdash G = \text{nf}_{\Gamma \vdash \kappa}(G) : \kappa$. Hence, we are done by corollary 2.2.3 and transitivity.

□

Now we can prove that a context Γ is equal to its normal form $\text{nf}(\Gamma)$. We need this statement for proving soundness of the subtyping algorithm.

Lemma 2.3.3 *Let Γ be a well defined context $\Gamma \vdash$. Then $\vdash \Gamma = \text{nf}(\Gamma)$.*

Proof. By induction on the inductive definition of Γ .

Case Γ equals \diamond . $\text{nf}(\diamond)$ is equal to \diamond , and $\vdash \diamond = \diamond$ by rule (EQC-EMPTY).

Case Γ equals $\Gamma', X \leq G : \kappa$ and $\text{nf}(\Gamma) = \text{nf}(\Gamma'), X \leq \text{nf}(G) : \kappa$. By induction hypothesis we have $\vdash \Gamma' = \text{nf}(\Gamma')$. Because of $\Gamma \vdash$ we have $\Gamma \vdash G : \kappa$ and also $\Gamma \vdash G = \text{nf}(G) : \kappa$ by lemma 2.3.2. So, by rule (EQC-BOUND) follows $\vdash \Gamma', X \leq G : \kappa = \text{nf}(\Gamma'), X \leq \text{nf}(G) : \kappa$. □

2.4. Characterization of long normal forms

If $\Gamma \vdash V \uparrow \kappa$ we say V is *hereditary normal* of kind κ . That means it is in η -long β -normal form and all *hidden redexes* are normalizable. For example the long normal form $\forall X \leq (\lambda Y.V) : * \rightarrow *. XW$ contains the hidden redex $(\lambda Y.V)W$.

Characterization of long normal forms. $\Gamma \vdash V \uparrow \kappa$. This judgement is designed such that the termination of the algorithm can be guaranteed. (s. next chapter).

$$\frac{(X \leq U : \vec{\kappa} \rightarrow *) \in \Gamma \quad \Gamma \vdash V_i \uparrow \kappa_i \text{ for all } i \quad \Gamma \vdash U @^{\vec{\kappa}} \vec{V} \uparrow *}{\Gamma \vdash X \vec{V} \uparrow *} \quad (\text{LN-VAR-BOUND})$$

$$\frac{\Gamma, X : \kappa \vdash V \uparrow \kappa'}{\Gamma \vdash \lambda X V \uparrow \kappa \rightarrow \kappa'} \quad (\text{LN-ABS}) \quad \frac{\Gamma \vdash V \uparrow * \quad \Gamma \vdash W \uparrow *}{\Gamma \vdash V \rightarrow W \uparrow *} \quad (\text{LN-ARROW})$$

$$\frac{\Gamma, X \leq U : \kappa \vdash V \uparrow *}{\Gamma \vdash \forall X \leq U : \kappa. V \uparrow *} \quad (\text{LN-ALL}) \quad \frac{\Gamma \uparrow}{\Gamma \vdash \top \uparrow *} \quad (\text{LN-TOP})$$

Admissible rules

$$\frac{\Gamma, \vec{X} : \vec{\kappa} \vdash V \uparrow \kappa'}{\Gamma \vdash \lambda \vec{X} V \uparrow \vec{\kappa} \rightarrow \kappa'} \quad (\text{LN-ABS-VECT})$$

Normal contexts $\Gamma \uparrow$.

$$\frac{}{\diamond \uparrow} \quad (\text{LNC-EMPTY}) \quad \frac{\Gamma \uparrow \quad \Gamma \vdash U \uparrow \kappa}{\Gamma, X \leq U : \kappa \uparrow} \quad (\text{LNC-BOUND})$$

Lemma 2.4.1 (Inversion) *If $\mathcal{D} :: \Gamma \vdash W \uparrow \kappa \rightarrow \kappa'$ then $W = \lambda X V$ and $\Gamma, X : \kappa \vdash V \uparrow \kappa'$.*

Proof. By looking at the rules we see that \mathcal{D} must be an instance of (LN-ABS), and we have by inversion: $\Gamma, X:\kappa \vdash V \uparrow \kappa'$. \square

Corollary 2.4.2 *If $\Gamma \vdash W \uparrow \vec{\kappa} \rightarrow *$ then $W = \lambda \vec{X}U$ and $\Gamma, \vec{X}:\vec{\kappa} \vdash U \uparrow *$.*

Proof. By induction on $|\vec{\kappa}|$.

Case $|\vec{\kappa}| = 0$

Nothing to show.

Case $|\vec{\kappa}| = n + 1$

So we have $\vec{\kappa} = \kappa, \vec{\kappa}_1$ with $|\vec{\kappa}_1| = n$ and we have $\Gamma \vdash W \uparrow \vec{\kappa} \rightarrow *$, i.e., $\Gamma \vdash W \uparrow \kappa \rightarrow \vec{\kappa}_1 \rightarrow *$. By inversion (Lemma 2.4.1) we get: $W = \lambda XV$ and $\Gamma, X:\kappa \vdash V:\vec{\kappa}_1 \rightarrow *$.

Now we can apply the induction hypothesis, and we get:

$V = \lambda \vec{X}U'$, hence, $W = \lambda X, \vec{X}.U'$ and $\Gamma, X:\kappa, \vec{X}:\vec{\kappa} \vdash U':*$. \square

Lemma 2.4.3 .

1. *If $\mathcal{D} :: \Gamma \vdash V \uparrow \kappa$ then $\Gamma \uparrow$.*
2. *If $\Gamma \uparrow$ and $(X \leq U : \kappa) \in \Gamma$ then $\Gamma \vdash U \uparrow \kappa$.*

Proof.

(1) By induction on \mathcal{D} .

Case (LN-TOP) We have $\Gamma \vdash \top \uparrow *$, and we show $\Gamma \uparrow$. But we have $\Gamma \uparrow$ by assumption.

Case (LN-VAR-BOUND) We have $\Gamma \vdash X\vec{V} \uparrow *$ and we show $\Gamma \uparrow$. We are done by the induction hypothesis.

Case (LN-ABS) We have $\Gamma \vdash \lambda XV \uparrow \kappa \rightarrow \kappa'$ and we show $\Gamma \uparrow$. By induction hypothesis, we have $\Gamma, X:\kappa \uparrow$, and by inversion, we get $\Gamma \uparrow$.

Case (LN-ARROW) We have $\Gamma \vdash V \rightarrow W \uparrow *$ and we show $\Gamma \uparrow$. We are finished by the induction hypothesis.

Case (LN-ALL) We have $\Gamma \vdash \forall X \leq U : \kappa. V \uparrow *$ and we show $\Gamma \uparrow$. By induction hypothesis we have $\Gamma, X \leq U : \kappa \uparrow$, and by inversion, we get $\Gamma \uparrow$.

(2) By induction on $\Gamma \uparrow$.

Case (LNC-EMPTY), impossible.

Case (LNC-BOUND) We have $\Gamma, Y \leq V : \kappa' \uparrow$ and $(X \leq U : \kappa) \in \Gamma$ and we show: $\Gamma, Y \leq V : \kappa' \vdash U \uparrow \kappa$. If $U = V$ then we have $\Gamma \vdash U \uparrow \kappa$ by assumption. Otherwise we have by induction hypothesis, $\Gamma \vdash U \uparrow \kappa$, and by weakening we conclude $\Gamma, Y \leq V : \kappa' \vdash U \uparrow \kappa$. \square

Lemma 2.4.4 *If $\mathcal{D} :: \Gamma, X \leq U : \kappa, \Gamma' \vdash V \uparrow \kappa'$ then $\mathcal{E} :: \Gamma \vdash U \uparrow \kappa$ and $\mathcal{E} \leq \mathcal{D}$.*

Now we prove the soundness of the characterization of terms in normal form. First we need to prove a technical lemma:

Lemma 2.4.5

1. *If $\Gamma \vdash V \uparrow \kappa$ and $\Gamma, Y : \kappa', \Gamma' \vdash Y : \kappa'$ then $[V^\kappa/Y]\text{nf}_{\Gamma \vdash \kappa'}(Y) \equiv V$.*
2. *If $\mathcal{D} :: \Gamma, X : \kappa, \Gamma' \vdash V \uparrow \kappa'$ then $[\text{nf}_{\Gamma \vdash \kappa}(X)^\kappa/X]V \equiv V$.*

Proof. Simultaneously by induction on κ .

1. *Case $\kappa = \vec{\kappa} \rightarrow *$.* We have $\Gamma \vdash V \uparrow \vec{\kappa} \rightarrow *$ and by inversion we have $\Gamma \vdash \lambda \vec{X}.V' \uparrow \vec{\kappa} \rightarrow *$ and we show

$$\begin{aligned}
[V^\kappa/Y]\text{nf}(Y) &\equiv V \text{ i.e.,} \\
[V^\kappa/Y](\lambda \vec{X}.Y \text{nf}(\vec{X})) &\equiv V \text{ i.e.,} \\
\lambda \vec{X}.V @^{\vec{\kappa}} \text{nf}(\vec{X}) &\equiv V \text{ i.e.,} \\
\lambda \vec{X}.(\lambda \vec{X}.V') @^{\vec{\kappa}} \text{nf}(\vec{X}) &\equiv \lambda \vec{X}.V' \text{ i.e.,} \\
\lambda \vec{X}.[\text{nf}(\vec{X})^{\vec{\kappa}}/\vec{X}]V' &\equiv \lambda \vec{X}.V'
\end{aligned}$$

We have by induction hypothesis (2): $[\text{nf}(X_i)^{\kappa_i}/X_i]V' \equiv V'$ for all i , thus, we are finished.

2. By local induction on \mathcal{D} . Let $\Gamma, X : \kappa, \Gamma' = \hat{\Gamma}$.

Case (LN-VAR-BOUND) We have $\hat{\Gamma} \vdash X \vec{V} \uparrow *$ and we show

$$\begin{aligned}
[\text{nf}(X)^\kappa/X](X \vec{V}) &\equiv X \vec{V} \text{ i.e.,} \\
\text{nf}(X) @^{\vec{\kappa}} [\text{nf}(X)^\kappa/X] \vec{V} &\equiv X \vec{V} \text{ i.e., by induction hypothesis (2)} \\
\text{nf}(X) @^{\vec{\kappa}} \vec{V} &\equiv X \vec{V} \text{ i.e.,} \\
(\lambda \vec{Y}.X \text{nf}(\vec{Y})) @^{\vec{\kappa}} \vec{V} &\equiv X \vec{V} \text{ i.e.,} \\
X [\vec{V}^{\vec{\kappa}}/\vec{Y}] \text{nf}(\vec{Y}) &\equiv X \vec{V} \text{ i.e.,}
\end{aligned}$$

Since $\kappa_i \leq \kappa$ for all i we have by induction hypothesis (1) $[V_i^{\kappa_i}/Y_i]\text{nf}(Y_i) = V_i$, thus, we are finished.

Case (LN-ABS) We have $\hat{\Gamma} \vdash \lambda Y V \uparrow \kappa_1 \rightarrow \kappa_2$ and we show

$$\begin{aligned}
[\text{nf}(X)^\kappa/X](\lambda Y V) &\equiv \lambda Y V \\
\lambda Y. [\text{nf}(X)^\kappa/X] V &\equiv \lambda Y V
\end{aligned}$$

We have by induction hypothesis (2) $[\text{nf}(X)^\kappa/X]V \equiv V$, thus, we are finished.

Case (LN-ARROW) We have $\hat{\Gamma} \vdash V \rightarrow W : *$ and we show

$$\begin{aligned}
[\text{nf}(X)^\kappa/X](V \rightarrow W) &\equiv V \rightarrow W \\
[\text{nf}(X)^\kappa/X]V \rightarrow [\text{nf}(X)^\kappa/X]W &\equiv V \rightarrow W
\end{aligned}$$

By induction hypothesis $[\text{nf}(X)^\kappa/X]V \equiv V$ and $[\text{nf}(X)^\kappa/X]W \equiv W$, thus, we are finished.

Case (LN-ALL) We have $\hat{\Gamma} \vdash \forall X \leq U : \kappa'. V \uparrow *$ and we show

$$\begin{aligned} [\text{nf}(X)^\kappa / X](\forall X \leq U : \kappa'. V) &\equiv \forall X \leq U : \kappa'. V \\ \forall X \leq [\text{nf}(X)^\kappa / X]U : \kappa'. [\text{nf}(X)^\kappa / X]V &\equiv \forall X \leq U : \kappa'. V \end{aligned}$$

By induction hypothesis $[\text{nf}(X)^\kappa / X]U \equiv U$ and $[\text{nf}(X)^\kappa / X]V \equiv V$, thus, we are finished.

Case (LN-TOP) We have $\hat{\Gamma} \vdash \top : *$ and we show $[\text{nf}(X)^\kappa / X]\top \equiv \top$ i.e., $\top \equiv \top$ which holds trivially. □

Lemma 2.4.6 (Soundness of the characterization of normal forms) *If $\mathcal{D} :: \Gamma \vdash U \uparrow \kappa$ then $\text{nf}(U) \equiv U$.*

Proof. By induction on \mathcal{D} .

Case (LN-VAR-BOUND) We have $\Gamma \vdash X\vec{V} \uparrow *$ and we show

$$\begin{aligned} \text{nf}(X\vec{V}) &\equiv X\vec{V} \text{ i.e.,} \\ \text{nf}(X) @^{\vec{\kappa}} \text{nf}(\vec{V}) &\equiv X\vec{V} \text{ i.e.,} \\ \lambda\vec{Y}. X \text{nf}(\vec{Y}) @^{\vec{\kappa}} \text{nf}(\vec{V}) &\equiv X\vec{V} \text{ i.e.,} \\ [\text{nf}(\vec{V})/\vec{Y}](X \text{nf}(\vec{Y})) &\equiv X\vec{V} \text{ i.e.,} \\ [\text{nf}(\vec{V})/\vec{Y}]X [\text{nf}(\vec{V})/\vec{Y}]\text{nf}(\vec{Y}) &\equiv X\vec{V} \text{ i.e.,} \\ X [\text{nf}(\vec{V})/\vec{Y}]\text{nf}(\vec{Y}) &\equiv X\vec{V} \text{ i.e., by induction hypothesis} \\ X [\vec{V}/\vec{Y}]\text{nf}(\vec{Y}) &\equiv X\vec{V} \end{aligned}$$

We have by assumption $\Gamma \vdash V_i \uparrow \kappa_i$ for all i . thus, we can apply lemma 2.4.5 (1) and we obtain $[V_i/Y_i]\text{nf}(Y_i) \equiv V_i$ for all i , and we are done.

Case (LN-ABS) We have $\Gamma \vdash \lambda XV \uparrow \kappa \rightarrow \kappa'$ and we show

$$\begin{aligned} \text{nf}_{\Gamma \vdash \kappa \rightarrow \kappa'}(\lambda XV) &= \lambda XV \text{ i.e.,} \\ \lambda X. \text{nf}_{\Gamma, X\kappa \vdash \kappa'}(V) &= \lambda XV \end{aligned}$$

But we have by induction hypothesis $\text{nf}_{\Gamma, X\kappa \vdash \kappa'}(V) = V$ thus, we are finished.

Case (LN-ARROW) We have $\Gamma \vdash V \rightarrow W : *$ and we show $\text{nf}_{\Gamma \vdash *}(V \rightarrow W) = V \rightarrow W$ i.e., $\text{nf}(V) \rightarrow \text{nf}(W) = V \rightarrow W$ but we have by induction hypothesis $\text{nf}(V) = V$ and $\text{nf}(W) = W$ thus, we are finished.

Case (LN-ALL) We have $\Gamma \vdash \forall X \leq U : \kappa. V \uparrow *$ and we show

$$\begin{aligned} \text{nf}_{\Gamma \vdash *}(\forall X \leq U : \kappa. V) &= \forall X \leq U : \kappa. V \text{ i.e.,} \\ \forall X \leq \text{nf}_{\Gamma \vdash \kappa}(U) : \kappa. \text{nf}_{\Gamma, X \leq U\kappa \vdash *}(V) &= \forall X \leq U : \kappa. V \end{aligned}$$

But we have by induction hypothesis $\text{nf}_{\Gamma \vdash \kappa}(U) = U$ and $\text{nf}_{\Gamma, X \leq U\kappa \vdash *}(V) = V$ thus, we are finished.

Case (LN-TOP) We have $\Gamma \vdash \top \uparrow *$ and we show $\text{nf}_{\Gamma \vdash *}(T) = T$ i.e., $T = T$ and this holds trivially. □

Now the goal is to show completeness, i.e., $\Gamma \vdash F : \kappa$ implies $\text{nf}(\Gamma) \vdash \text{nf}(F) \uparrow \kappa$.

The following lemma is the heart of our technical development; it simplifies the metatheory considerably. In contrast to previous approaches, we do not need to construct a model of constructors [CG03] or induct on maximal reduction lengths [Gog05].

Lemma 2.4.7 (Substitution and application for normal forms) *Let $\mathcal{E} :: \Gamma \vdash U \uparrow \kappa$.*

1. *Assume $\mathcal{D} :: \Gamma, X : \kappa, \Gamma' \uparrow$. Then $\Gamma, [U^\kappa/X]\Gamma' \uparrow$.*
2. *Assume $\mathcal{D} :: \Gamma, X : \kappa, \Gamma' \vdash V \uparrow \kappa'$. Then $\Gamma, [U^\kappa/X]\Gamma' \vdash [U^\kappa/X]V \uparrow \kappa'$.*
3. *Assume $\mathcal{D} :: \Gamma \vdash V \uparrow \kappa \rightarrow \kappa'$. Then $\Gamma \vdash V @^\kappa U \uparrow \kappa'$.*

Proof. Simultaneously by lexicographic induction on (κ, \mathcal{D}) .

1.

Case (LNC-EMPTY) impossible.

Case (LNC-BOUND) We have $\Gamma, X : \kappa, \Gamma', Y \leq W : \kappa' \uparrow$ and we show $\Gamma, [U^\kappa/X](\Gamma', Y \leq W : \kappa') \uparrow$ i.e., $\Gamma, [U^\kappa/X]\Gamma', Y \leq [U^\kappa/X]W : \kappa' \uparrow$. By induction hypothesis (1) we have: $\Gamma, [U^\kappa/X]\Gamma' \uparrow$, and by induction hypothesis (2) we have: $\Gamma, [U^\kappa/X]\Gamma' \vdash [U^\kappa/X]W \uparrow \kappa$. So we can finish with (LNC-BOUND).

2.

Case (LN-TOP) We have $\Gamma, X : \kappa, \Gamma' \vdash \top \uparrow *$ and we show $\Gamma, [U^\kappa/X]\Gamma' \vdash [U^\kappa/X]\top \uparrow *$ i.e., $\Gamma, [U^\kappa/X]\Gamma' \vdash \top \uparrow *$. We have by induction hypothesis (1): $\Gamma, [U^\kappa/X]\Gamma' \uparrow$, thus, we can finish with (LN-TOP).

Case (LN-ABS) We have $\Gamma, X : \kappa, \Gamma' \vdash \lambda Y V \uparrow \kappa_1 \rightarrow \kappa_2$ and we show: $\Gamma, [U^\kappa/X]\Gamma' \vdash [U^\kappa/X](\lambda Y V) \uparrow \kappa_1 \rightarrow \kappa_2$, i.e., $\Gamma, [U^\kappa/X]\Gamma' \vdash \lambda Y [U^\kappa/X]V \uparrow \kappa_1 \rightarrow \kappa_2$ w.l.o.g. $Y \neq X$ and $Y \notin FV(U)$. We have by induction hypothesis, $\Gamma, [U^\kappa/X]\Gamma', Y : \kappa_1 \vdash [U^\kappa/X]V \uparrow \kappa_2$, thus, we can apply (LN-ABS) and we are done.

Case (LN-ARROW) We have $\Gamma, X : \kappa, \Gamma' \vdash V \rightarrow W \uparrow *$ and we show: $\Gamma, [U^\kappa/X]\Gamma' \vdash [U^\kappa/X](V \rightarrow W) \uparrow *$ i.e., $\Gamma, [U^\kappa/X]\Gamma' \vdash [U^\kappa/X]V \rightarrow [U^\kappa/X]W \uparrow *$. By induction hypothesis holds $\Gamma, [U^\kappa/X]\Gamma' \vdash [U^\kappa/X]V \uparrow *$ and $\Gamma, [U^\kappa/X]\Gamma' \vdash [U^\kappa/X]W \uparrow *$, thus, we can finish by (LN-ARROW).

Case (LN-ALL) We have $\Gamma, X : \kappa, \Gamma' \vdash \forall Y \leq W : \kappa_1. V \uparrow *$ and we show: $\Gamma, [U^\kappa/X]\Gamma' \vdash [U^\kappa/X](\forall Y \leq W : \kappa_1. V) \uparrow *$, i.e., $\Gamma, [U^\kappa/X]\Gamma' \vdash \forall Y \leq [U^\kappa/X]W : \kappa_1. [U^\kappa/X]V \uparrow *$ w.l.o.g. $Y \neq X$ and $Y \notin FV(U)$. By induction hypothesis, we have $\Gamma, [U^\kappa/X]\Gamma', Y \leq [U^\kappa/X]W : \kappa_1 \vdash [U^\kappa/X]V \uparrow *$, thus, we can finish with (LN-ALL).

Case (LN-VAR-BOUND) We have $\Gamma, X : \kappa, \Gamma' \vdash Y \vec{V} \uparrow *$ and there is a W with $Y \leq W : \kappa' \in \Gamma, X : \kappa, \Gamma'$ and we show $\Gamma, [U^\kappa/X]\Gamma \vdash [U^\kappa/X](Y \vec{V})$. $\kappa' = \vec{\kappa} \rightarrow *$.

Subcase $X = Y$, then we have: $\Gamma, X \leq \top_\kappa : \kappa, \Gamma' \vdash X\vec{V} \uparrow *$ and we show: $\Gamma, [U^\kappa/X]\Gamma' \vdash [U^\kappa/X](X\vec{V}) \uparrow *$ which is equivalent to

$$\overbrace{\Gamma, [U^\kappa/X]\Gamma'}^{\Gamma''} \vdash U @^{\vec{\kappa}} \overbrace{[U^\kappa/X]\vec{V}}^{\vec{V}'} \uparrow *$$

By induction hypothesis (2), we have: $\mathcal{D}_i :: \Gamma'' \vdash V'_i \uparrow \kappa_i$ because $(\kappa, \mathcal{D}_i) < (\kappa, \mathcal{D})$.

Moreover, we have by induction hypothesis (3):

$$\mathcal{E}_1 :: \Gamma'' \vdash U @^{\kappa_1} V'_1 \uparrow \kappa_2 \rightarrow \dots \rightarrow \kappa_n \rightarrow * \text{ because } (\kappa_1, \mathcal{E}) < (\kappa, \mathcal{D})$$

We also have by induction hypothesis (3):

$$\mathcal{E}_2 :: \Gamma'' \vdash U @^{\kappa_1, \kappa_2} V'_1 V'_2 \uparrow \kappa_3 \rightarrow \dots \rightarrow \kappa_n \rightarrow * \text{ because } (\kappa_2, \mathcal{E}_1) < (\kappa, \mathcal{D})$$

⋮

$$\mathcal{E}_n :: \Gamma'' \vdash U @^{\vec{\kappa}} \vec{V}' \uparrow *$$

and we are done.

Subcase $X \neq Y$,

$$\frac{(\Gamma, X : \kappa, \Gamma' \vdash V_i \uparrow \kappa_i)_i \quad (Y \leq W : \kappa') \in (\Gamma, X : \kappa, \Gamma') \quad \Gamma, X : \kappa, \Gamma' \vdash W @^{\vec{\kappa}} \vec{V} \uparrow *}{\Gamma, X : \kappa, \Gamma' \vdash Y\vec{V} \uparrow *}$$

We have: $\Gamma, X : \kappa, \Gamma' \vdash Y\vec{V} \uparrow *$ and there is a W with $Y \leq W : \kappa' \in \Gamma, X : \kappa, \Gamma'$ and we show: $\Gamma, [U^\kappa/X]\Gamma' \vdash [U^\kappa/X](Y\vec{V}) \uparrow *$ which is equivalent to

$$\overbrace{\Gamma, [U^\kappa/X]\Gamma'}^{\Gamma''} \vdash Y \overbrace{[U^\kappa/X]\vec{V}}^{\vec{V}'} \uparrow *$$

By induction hypothesis (2), we have: $\Gamma'' \vdash V'_i \uparrow \kappa_i$ for all $i = 1 \dots |\kappa|$, and also by induction hypothesis (2) we have $\Gamma'' \vdash [U^\kappa/X](W @^{\vec{\kappa}} \vec{V})$, i.e. by lemma 2.2.4, $\Gamma'' \vdash [U^\kappa/X]W @^{\vec{\kappa}} \vec{V}'$.

So we can apply (LN-VAR-BOUND) and we get:

$$\frac{\Gamma'' \vdash V'_i \uparrow \kappa_i \text{ for all } i \quad \Gamma'' \vdash [U^\kappa/X]W @^{\vec{\kappa}} \vec{V}' \uparrow *}{\Gamma'' \vdash Y\vec{V}' \uparrow *} \text{ (LN-VAR-BOUND)}$$

3. The only possible case is (LN-ABS). We have $\Gamma \vdash \lambda X V' \uparrow \kappa \rightarrow \kappa'$ and we show: $\Gamma \vdash \lambda X V' @^\kappa U \uparrow \kappa'$ which is equivalent to $\Gamma \vdash [U^\kappa/X]V' \uparrow \kappa'$ and it follows from part (2).

□

Lemma 2.4.8 *If $X \leq G : \kappa \in \Gamma$ and $\text{nf}(\Gamma) \vdash \text{nf}(G) \uparrow \kappa$ then $\text{nf}(\Gamma) \vdash \text{nf}(X) \uparrow \kappa$.*

Proof. By induction on κ .

Let $\kappa = \vec{\kappa} \rightarrow *$ and $\text{nf}(\Gamma) = \Gamma'$ and $\text{nf}(G) = U$. We show $\Gamma' \vdash \text{nf}_{\Gamma \vdash \vec{\kappa} \rightarrow *} (X) \uparrow \kappa$, i.e., $\Gamma' \vdash \lambda Y_1 \dots \lambda Y_n. X \text{nf}_{\Gamma, \vec{\kappa} \vdash \kappa_1} (Y_1) \dots \text{nf}_{\Gamma, \vec{\kappa} \vdash \kappa_n} (Y_n) \uparrow \kappa$. We have by induction hypothesis: $\Gamma', \vec{Y} : \vec{\kappa} \vdash \text{nf}(Y_i) \uparrow \kappa_i$, and by corollary 2.4.2 it holds that: $U = \lambda \vec{Y} U'$ and $\Gamma', \vec{Y} : \vec{\kappa} \vdash U' \uparrow *$. Notice that by renaming we may assume the same variable names in both claims. Moreover we have $U @^\kappa \text{nf}(\vec{Y}) = [(\text{nf}(\vec{Y}))^\kappa / \vec{Y}] U' = U'$ by lemmag 2.4.5, part (2), thus, we can derive:

$$\frac{(X \leq U : \vec{\kappa} \rightarrow *) \in \Gamma' \quad \Gamma', \vec{Y} : \vec{\kappa} \vdash \text{nf}(Y_i) \uparrow \kappa_i \quad \Gamma', \vec{Y} : \vec{\kappa} \vdash U @^\kappa \text{nf}(\vec{Y}) \uparrow *}{\frac{\Gamma', \vec{Y} : \vec{\kappa} \vdash X \text{nf}(\vec{Y}) \uparrow *}{\Gamma' \vdash \lambda \vec{Y}. X \text{nf}(\vec{Y}) \uparrow \vec{\kappa} \rightarrow *} \text{ (LN-ABS-VECT)}}$$

□

Theorem 2.4.9 (Completeness of the characterization of normal forms)

1. If $\mathcal{D} :: \Gamma \vdash$ then $\text{nf}(\Gamma) \uparrow$.
2. If $\mathcal{D} :: \Gamma \vdash F : \kappa$ then $\text{nf}(\Gamma) \vdash \text{nf}(F) \uparrow \kappa$.

Proof. Simultaneously by induction on \mathcal{D} .

1. *Case* (C-EMPTY) We have $\diamond \vdash$ and we show $\diamond \uparrow$. It follows with (LNC-EMPTY).
Case (C-BOUND) We have $\Gamma, X \leq G : \kappa \vdash$ and we show $\Gamma, X \leq G : \kappa \uparrow$. By induction hypothesis (2) we have $\text{nf}(\Gamma) \vdash \text{nf}(G) \uparrow \kappa$ and by induction hypothesis (1) we have $\Gamma \uparrow$ thus, we can finish with (LNC-BOUND).
2. *Case* (K-VAR-BOUND) We have $\Gamma \vdash X : \kappa$ and we show $\text{nf}(\Gamma) \vdash \text{nf}(X) : \kappa$. It follows by lemma 2.4.8.
Case (K-TOP) We have $\Gamma \vdash \top : *$ and we show $\text{nf}(\Gamma) \vdash \text{nf}(\top) \uparrow *$ i.e., $\text{nf}(\Gamma) \vdash \top \uparrow *$. We have by induction hypothesis (1) $\text{nf}(\Gamma) \uparrow$, thus, we can finish with (LN-TOP).
Case (K-ABS) We have $\Gamma \vdash \lambda X V : \kappa \rightarrow \kappa'$ and we show $\text{nf}(\Gamma) \vdash \text{nf}(\lambda X V) \uparrow \kappa \rightarrow \kappa'$. We have by induction hypothesis (2): $\text{nf}(\Gamma), X : \kappa \vdash \text{nf}(V) \uparrow \kappa'$, thus, we can finish by (LN-ABS).
Case (K-APP) We have $\Gamma \vdash FG : \kappa'$ and we show: $\text{nf}(\Gamma) \vdash \text{nf}(FG) \uparrow \kappa'$, i.e., $\text{nf}(\Gamma) \vdash \text{nf}(F) @^\kappa \text{nf}(G) \uparrow \kappa'$. By induction hypothesis (2) holds: $\text{nf}(\Gamma) \vdash \text{nf}(F) \uparrow \kappa \rightarrow \kappa'$ and $\text{nf}(\Gamma) \vdash \text{nf}(G) \uparrow \kappa$. Thus, we can apply the lemma substitution and application for normal forms (Lemma 2.4.7), part 3, which concludes the proof.
Case (K-ARROW) We have $\Gamma \vdash A \rightarrow B : *$ and we show: $\text{nf}(\Gamma) \vdash \text{nf}(A \rightarrow B) \uparrow *$, i.e., $\text{nf}(\Gamma) \vdash \text{nf}(A) \rightarrow \text{nf}(B) \uparrow *$. By induction hypothesis (2) we have $\text{nf}(\Gamma) \vdash \text{nf}(A) \uparrow * \text{nf}(\Gamma) \vdash \text{nf}(B) \uparrow *$, thus, we finish by (LN-ARROW).
Case (K-ALL) We have $\Gamma \vdash \forall X \leq G : \kappa. A : *$ and we show: $\text{nf}(\Gamma) \vdash \text{nf}(\forall X \leq G : \kappa. A) \uparrow *$, i.e., $\text{nf}(\Gamma) \vdash \forall X \leq \text{nf}(G) : \kappa. \text{nf}(A) \uparrow *$. We have by induction hypothesis (2): $\text{nf}(\Gamma), X \leq \text{nf}(G) : \kappa \vdash \text{nf}(A) \uparrow *$, thus, we can finish by (LN-ALL).

□

2.5. Completeness of normalization

We have defined a normalization function for type constructors and shown its soundness. Afterwards we have defined an inductive characterization of type constructors in normal form, and proved its soundness and completeness as well. In this section, we will use that characterization for proving the most important property of the normalization function, its completeness, i.e., that the normal forms of two equal terms are identical.

The proof is based on the following lemma, which captures the intuition that hereditary substitution preserves normal forms.

Lemma 2.5.1 *If $\mathcal{D} :: \Gamma, X : \kappa, \Gamma' \vdash F : \kappa'$ and $\Gamma \vdash G : \kappa$. then $\text{nf}([G/X]F) \equiv [(\text{nf}(G))^\kappa/X]\text{nf}(F)$.*

Proof. By induction on \mathcal{D} .

Case (K-VAR) We have $\Gamma, X : \kappa, \Gamma' \vdash Y : \kappa'$ and we show $\text{nf}([G/X]Y) \equiv [(\text{nf}(G))^\kappa/X]\text{nf}(Y)$.

Subcase $X = Y$ Then we have $\Gamma, X : \kappa \vdash X : \kappa$ and we show

$$\begin{aligned} \text{nf}([G/X]X) &\equiv [(\text{nf}(G))^\kappa/X]\text{nf}(X) \text{ i.e.,} \\ \text{nf}(G) &\equiv [(\text{nf}(G))^\kappa/X]\text{nf}(X) \end{aligned}$$

By theorem 2.4.9 we have $\text{nf}(G) \uparrow \kappa$, thus, we can apply the lemma 2.4.5 part (1) and we are done.

Subcase $X \neq Y$ Then we have to show

$$\begin{aligned} \text{nf}([G/X]Y) &\equiv [(\text{nf}(G))^\kappa/X]\text{nf}(Y) \text{ i.e.,} \\ \text{nf}(Y) &\equiv [(\text{nf}(G))^\kappa/X]\lambda\vec{Y}.Y\text{nf}(\vec{Y}) \text{ i.e.,} \\ \text{nf}(Y) &\equiv \lambda\vec{Y}.Y\text{nf}(\vec{Y}) \end{aligned}$$

And that holds by definition.

Case (K-TOP) We have $\Gamma, X : \kappa \vdash \top : *$ and $\Gamma \vdash G : \kappa$ and we show $\text{nf}([G/X]\top) \equiv [(\text{nf}(G))^\kappa/X]\text{nf}(\top)$ i.e., $\top \equiv \top$ which holds trivially.

Case (K-ABS) We have $\Gamma, X : \kappa, \Gamma' \vdash \lambda Y F : \kappa_1 \rightarrow \kappa_2$ and $\Gamma \vdash G : \kappa$ and we show $\text{nf}([G/X]\lambda Y F) \equiv [(\text{nf}(G))^\kappa/X]\text{nf}(\lambda Y F)$ i.e., $\lambda Y \text{nf}([G/X]F) \equiv \lambda Y [(\text{nf}(G))^\kappa/X]\text{nf}(F)$. By induction hypothesis we have $\text{nf}([G/X]F) \equiv [(\text{nf}(G))^\kappa/X]\text{nf}(F)$ which concludes the proof.

Case (K-ARROW) We have $\Gamma, X : \kappa, \Gamma' \vdash A \rightarrow B : *$ and $\Gamma \vdash G : \kappa$ and we show $\text{nf}([G/X](A \rightarrow B)) \equiv [(\text{nf}(G))^\kappa/X]\text{nf}(A \rightarrow B)$ i.e., $\text{nf}([G/X]A) \rightarrow \text{nf}([G/X]B) \equiv [(\text{nf}(G))^\kappa/X]\text{nf}(A) \rightarrow [(\text{nf}(G))^\kappa/X]\text{nf}(B)$. By induction hypothesis we have $\text{nf}([G/X]A) \equiv [(\text{nf}(G))^\kappa/X]\text{nf}(A)$ and $\text{nf}([G/X]B) \equiv [(\text{nf}(G))^\kappa/X]\text{nf}(B)$ which conclude the proof.

Case (K-ALL) We have $\Gamma, X : \kappa \vdash \forall Y \leq H : \kappa. A : *$ and $\Gamma \vdash G : \kappa$ and we show

$$\text{nf}([G/X](\forall Y \leq H : \kappa. A)) \equiv [(\text{nf}(G))^\kappa/X]\text{nf}(\forall Y \leq H : \kappa. A) : * \text{ i.e.,}$$

$$\forall Y \leq \text{nf}([G/X]H) : \kappa. \text{nf}([G/X]A) \equiv \forall Y \leq [(\text{nf}(G))^\kappa/X]\text{nf}(H) : \kappa. [(\text{nf}(G))^\kappa/X]\text{nf}(A)$$

By induction hypothesis we have: $\text{nf}([G/X]A) \equiv [(\text{nf}(G))^\kappa/X]\text{nf}(A)$ and $\text{nf}([G/X]H) \equiv [(\text{nf}(G))^\kappa/X]\text{nf}(H)$ which concludes the proof.

Case (K-APP) We have $\Gamma, X:\kappa, \Gamma' \vdash FH:\kappa_2$ and $\Gamma \vdash G:\kappa$ and, by inversion, $\Gamma, X:\kappa, \Gamma' \vdash F:\kappa_1 \rightarrow \kappa_2$ and we show

$$\begin{aligned} \text{nf}([G/X](FH)) &\equiv [(\text{nf}(G))^\kappa/X]\text{nf}(FH), \text{ i.e.,} \\ \text{nf}([G/X]F) @^{\kappa_1} \text{nf}([G/X]H) &\equiv [(\text{nf}(G))^\kappa/X](\text{nf}(F) @^{\kappa_1} \text{nf}(H)) \text{ i.e., by Lemma 2.2.4} \\ \text{nf}([G/X]F) @^{\kappa_1} \text{nf}([G/X]H) &\equiv [(\text{nf}(G))^\kappa/X]\text{nf}(F) @^{\kappa_1} [(\text{nf}(G))^\kappa/X]\text{nf}(H) \end{aligned}$$

which follows by the induction hypothesis. \square

Now the theorem can be proved by an easy induction.

Theorem 2.5.2 (Completeness of the normalization) *If $\mathcal{D} :: \Gamma \vdash F = F' : \kappa$ then $\text{nf}(F) \equiv \text{nf}(F')$.*

Proof. By induction on \mathcal{D} .

Case (EQ- β) We have $\Gamma \vdash (\lambda XF)G = [G/X]F:\kappa$ and we show $\text{nf}((\lambda XF)G) \equiv \text{nf}([G/X]F)$ i.e., $[(\text{nf}(G))^\kappa/X]\text{nf}(F) \equiv \text{nf}([G/X]F)$, and this follows by lemma 2.5.1.

Case (EQ- η) We have $\Gamma \vdash (\lambda X.FX) = F:\kappa \rightarrow \kappa'$ and we show $\text{nf}(\lambda X.FX) \equiv \text{nf}(F)$ i.e., $\lambda X.\text{nf}(F) @^\kappa \text{nf}(X) \equiv \text{nf}(F)$. We have by theorem 2.4.9 $\Gamma \vdash \text{nf}(F) \uparrow \kappa \rightarrow \kappa'$, thus, by inversion, since w.l.o.g. $X \notin \text{FV}(F)$, $F = \lambda X.V$. Moreover, $F = \lambda X.\text{nf}(V)$, since $V \equiv \text{nf}(V)$ by lemma 2.4.6. Then we show:

$$\begin{aligned} \lambda X.(\lambda X.\text{nf}(V)) @^\kappa \text{nf}(X) &\equiv \lambda X.\text{nf}(V) \text{ i.e.,} \\ \lambda X.[(\text{nf}(X))^\kappa/X]\text{nf}(V) &\equiv \lambda X.\text{nf}(V) \text{ i.e., by lemma 2.5.1} \\ \lambda X.\text{nf}([X/X]V) &\equiv \lambda X.\text{nf}(V) \text{ i.e.,} \\ \lambda X.\text{nf}(V) &\equiv \lambda X.\text{nf}(V) \end{aligned}$$

Case (EQ-VAR) We have $\Gamma \vdash X = X:\kappa$ and we show $\text{nf}(X) \equiv \text{nf}(X)$ which is trivially true.

Case (EQ-TOP) We have $\Gamma \vdash \top = \top:*$ and we show $\text{nf}(\top) \equiv \text{nf}(\top)$ which is trivially true.

Case (EQ- λ) We have $\Gamma \vdash \lambda XF = \lambda XF':\kappa \rightarrow \kappa'$ and we show $\text{nf}(\lambda XF) \equiv \text{nf}(\lambda XF')$, i.e., $\lambda X.\text{nf}(F) \equiv \lambda X.\text{nf}(F')$. By induction hypothesis we have $\text{nf}(F) \equiv \text{nf}(F')$, and we are finished.

Case (EQ- \forall) We have $\Gamma \vdash \forall X \leq G:\kappa.A = \forall X \leq G':\kappa.A':*$ and we show $\text{nf}(\forall X \leq G:\kappa.A) \equiv \text{nf}(\forall X \leq G':\kappa.A')$, i.e., $\forall X \leq \text{nf}(G):\kappa.\text{nf}(A) \equiv \forall X \leq \text{nf}(G'):\kappa.\text{nf}(A')$. By induction hypothesis we have $\text{nf}(G) \equiv \text{nf}(G')$ and $\text{nf}(A) \equiv \text{nf}(A')$, thus, we are finished.

Case (EQ-ARROW) We have $\Gamma \vdash A \rightarrow B = A' \rightarrow B':*$ and we show $\text{nf}(A \rightarrow B) \equiv \text{nf}(A' \rightarrow B')$ i.e., $\text{nf}(A) \rightarrow \text{nf}(B) \equiv \text{nf}(A') \rightarrow \text{nf}(B')$. By induction hypothesis we have $\text{nf}(A) \equiv \text{nf}(A')$ and $\text{nf}(B) \equiv \text{nf}(B')$, thus, we are finished.

Case (EQ-APP) We have $\Gamma \vdash FG = F'G':\kappa'$ and we show $\text{nf}(FG) \equiv \text{nf}(F'G')$ i.e., $\text{nf}(F) @^\kappa \text{nf}(G) \equiv \text{nf}(F') @^\kappa \text{nf}(G')$. By induction hypothesis we have $\text{nf}(F) \equiv \text{nf}(F')$ and $\text{nf}(G) \equiv \text{nf}(G')$, thus, we are finished.

Case (EQ-SYM) We have $\Gamma \vdash F' = F : \kappa$ and we show $\text{nf}(F') \equiv \text{nf}(F)$. By induction hypothesis we have $\text{nf}(F) \equiv \text{nf}(F')$, thus, we are finished.

Case (EQ-TRANS) We have $\Gamma \vdash F_1 = F_3 : \kappa$ and we show $\text{nf}(F_1) \equiv \text{nf}(F_3)$. By induction hypothesis we have $\text{nf}(F_1) \equiv \text{nf}(F_2)$ and $\text{nf}(F_2) \equiv \text{nf}(F_3)$, thus, we are finished by transitivity of the syntactical equality. \square

We get as a corollary the uniqueness of normal forms.

Corollary 2.5.3 (Uniqueness of normal forms) *If $\Gamma \vdash V = V' : \kappa$ and $\Gamma \vdash V, V' \uparrow \kappa$ then $V \equiv V'$.*

Proof. We have by theorem 2.5.2, $\text{nf}(V) \equiv \text{nf}(V')$. Moreover, we have $V \equiv \text{nf}(V)$ and $V' \equiv \text{nf}(V')$ by lemma 2.4.6, thus, by transitivity of the syntactical equality, we have $V \equiv \text{nf}(V) \equiv \text{nf}(V') \equiv V'$. \square

2.6. Summary

We give a short summary of the results of this chapter.

- $\Gamma \vdash F = F' : \kappa$ can be decided by checking $\text{nf}(F) \equiv \text{nf}(F')$.
- If $\Gamma \vdash F : \kappa$ then $\text{nf}(\Gamma) \vdash \text{nf}(F) \uparrow \kappa$.
- $\text{nf}()$ is idempotent: $\text{nf}(\text{nf}(F)) \equiv F$ because $\text{nf}(V) \equiv V$ for $\Gamma \vdash V \uparrow \kappa$.

We have achieved many useful results in this chapter. Some of them will be needed in the next chapter, when we prove soundness, completeness, and termination of the algorithm for deciding subtyping.

Soundness. For soundness of the algorithm, we need the results related to $\text{nf}(\Gamma)$ and with equality of contexts, i.e., $\vdash \Gamma = \text{nf}(\Gamma)$ and equal contexts preserve kinding, equality, and subtyping.

Completeness. For showing completeness we need the completeness of the normalization function, i.e., $\Gamma \vdash F = F' : \kappa$ implies $\text{nf}(F) \equiv \text{nf}(F')$.

Termination. We will prove that the algorithm terminates when it is applied to hereditary normal constructors. Hence the completeness of the characterization of constructors in normal form is necessary, i.e., $\Gamma \vdash F : \kappa$ implies $\text{nf}(\Gamma) \vdash \text{nf}(F) \uparrow \kappa$.

3. Algorithmic Subtyping

The usual way of stating decidability of subtyping of system F_{\leq}^{ω} : (and of similar systems) is the following:

- Starting with a declarative presentation of the system that directly expresses its intended meaning, but which is not directly implementable (s. Chapter 1).
- Proposing an alternative presentation of the same relation by a *syntax directed* set of inference rules. This means that the system can be implemented by a proof-search semi-algorithm that will never have to guess or backtrack.
- Check that this semi-algorithm is indeed an algorithm by showing that proof-search must terminate in finite time when starting with any statement as its initial goal.
- Show that the syntax-directed system is sound, in the sense that any subtyping statement derived by the algorithm is also derivable in the original system. This step is typically straightforward.
- Finally, prove that the syntax-directed system is complete: that any statement derivable in the original system is also derivable by the algorithm. This step is usually the most difficult one.

Soundness and completeness together mean that the declarative and algorithmic system are equivalent, thus, the implementation of the syntax-directed rules is a decision procedure for the original system.

The syntax-directed system may be viewed as a version of the original system, from which all problematic rules have been removed. One of these rules is (S-TRANS):

$$\frac{\Gamma \vdash F \leq G : \kappa \quad \Gamma \vdash G \leq H : \kappa}{\Gamma \vdash F \leq H : \kappa} \text{ (S-TRANS)}$$

This rule is problematic because when deciding $\Gamma \vdash F \leq H : \kappa$ the algorithm would need to somehow guess a G , so that the premises of the rule hold. Fortunately, in most cases this rule can be eliminated by rewriting derivations. There is one case, though, where transitivity is indeed necessary. Statements with variables on the left-hand side cannot, in general, be proved without using transitivity. We saw already an example in Chapter 1:

$$X \leq \text{ResetCounterM} : * \rightarrow * \vdash X \leq \text{CounterM} : * \rightarrow *$$

It was proved using an instance of (S-VAR) to establish the connection between X and ResetCounterM . We had proved before $\Gamma \vdash \text{ResetCounterM} \leq \text{CounterM} : * \rightarrow *$, thus, we could finish with an instance of transitivity. Thus, to eliminate (S-TRANS) while retaining completeness, it is necessary to refine the treatment of variables, extending each instance of (S-VAR) with an internal use of transitivity:

$$\frac{(X \leq G:\kappa) \quad \Gamma \vdash G \leq H:\kappa}{\Gamma \vdash X \leq H:\kappa} \text{ temp. rule 1}$$

Let us look at another kind of situation in which transitivity plays an essential role. In the context $\Gamma = Y \leq \lambda X X:*\rightarrow*$, the statement $\Gamma \vdash Y A \leq A:*$ is provable as follows:

$$\frac{\frac{\frac{}{\Gamma \vdash Y \leq \lambda X X:*\rightarrow*} \text{ (S-VAR)}}{\Gamma \vdash Y A \leq (\lambda X X)A:*\text{ (S-APP)}} \quad \frac{}{\Gamma \vdash (\lambda X X)A \leq A:*\text{ (S-EQ)}}}{\Gamma \vdash Y A \leq A:*\text{ (S-TRANS)}}$$

The instance of transitivity in the derivation is again essential, but it is not an instance of the previous schema. In fact, it is possible to construct more involved examples where the instance of (S-VAR) is separated from the instance of (S-TRANS) by arbitrarily many applications of (S-APP). The solution to this problem is to extend the temporary rule 1 as follows:

$$\frac{(X \leq G:\kappa) \quad \Gamma \vdash G\vec{F} \leq H:*\text{ temp. rule 2}}{\Gamma \vdash X\vec{F} \leq H:*$$

Moreover, we need to deal with the possibility of conversion. We need to perform some reduction on the argument before choosing which clause of the algorithm to apply. The easiest way is to normalize the arguments before looking at them. For this purpose we defined a normalizer in the last chapter. The only rule of the algorithm that requires a renormalization of the constructors is the bound rule. Actually, we do not need to completely normalize the constructors again, but we only need to eliminate the new redexes that appear in the case that the bound is a lambda abstraction. Thus, if we use the function @ from last chapter, we preserve the normal form of the constructors. This motivates the definitive rule (SA-BOUND):

$$\frac{(X \leq U) \in \Gamma \quad \Gamma \vdash_a U @^{\vec{\kappa}} \vec{V} \leq W:*\text{ (SA-BOUND)}}{\Gamma \vdash_a X\vec{V} \leq W:*$$

The rule (S-EQ) relates equal constructors. Since we apply the algorithmic rules to constructors in normal form, and the normal forms of equal constructors are identical, we replace the rule (S-EQ) with (SA-REFL), which compares syntactically equal constructors of the form $X\vec{V}$. Actually, the only application in normal forms is $X\vec{V}$, hence, we remove the rule (S-APP). The other subtyping rules remain unchanged. We obtain the following syntax-directed system:

Algorithmic subtyping for higher-order bounded quantification ($F_{<}^\omega$)

$$\begin{array}{c}
\frac{}{\Gamma \vdash_a X\vec{V} \leq X\vec{V} : *} \text{ (SA-REFL)} \quad \frac{}{\Gamma \vdash_a V \leq \top : *} \text{ (SA-TOP)} \\
\\
\frac{(X \leq U : \kappa) \in \Gamma \quad \Gamma \vdash_a U @^{\vec{\kappa}} \vec{V} \leq W : *}{\Gamma \vdash_a X\vec{V} \leq W : *} \text{ (SA-BOUND)} \\
\\
\frac{\Gamma, (X_i \leq \top_{\kappa_i} : \kappa_i) \vdash_a V \leq V' : *}{\Gamma \vdash_a \lambda \vec{X}. V \leq \lambda \vec{X}. V' : \vec{\kappa} \rightarrow *} \text{ (SA-ABS)} \quad \frac{\Gamma \vdash_a W \leq V : * \quad \Gamma \vdash_a V' \leq W' : *}{\Gamma \vdash_a V \rightarrow V' \leq W \rightarrow W' : *} \text{ (SA-ARROW)} \\
\\
\frac{\Gamma, X \leq V : \kappa \vdash_a W \leq W' : *}{\Gamma \vdash_a \forall X \leq V : \kappa. W \leq \forall X \leq V : \kappa. W' : *} \text{ (SA-ALL)}
\end{array}$$

In the next sections we show that this algorithm is sound and complete for the declarative rules in Chapter 1. In the following lemmas, we use the characterization of normal forms defined in last chapter $\Gamma \vdash V \uparrow \kappa$, and the fact that every normal term can be characterized by this judgement.

3.1. Soundness of algorithmic subtyping

The soundness of the algorithm is relatively easy to prove because the algorithmic rules are less permissive as the declarative ones. The proof is an easy induction on derivations, and we use the statements from Chapter 2: equal contexts preserve equality and subtyping, and $\vdash \Gamma = \text{nf}(\Gamma)$.

Lemma 3.1.1 *Let $\Gamma \vdash V, V' : \kappa$. If $\mathcal{D} :: \Gamma \vdash_a V \leq V' : \kappa$, then $\Gamma \vdash V \leq V' : \kappa$.*

Proof. By induction on \mathcal{D} .

Case (SA-REFL). We have $V = V' = X\vec{W}$. It follows by rule (S-EQ).

Case (SA-TOP). We have $V' = \top$. It follows with rule (S-TOP).

Case (SA-BOUND). $\frac{(X \leq U : \kappa) \in \Gamma \quad \Gamma \vdash_a U @^{\vec{\kappa}} \vec{V} \leq W : *}{\Gamma \vdash_a X\vec{V} \leq W : *} \text{ (SA-BOUND)}$

We have $V = X\vec{V}$ and $V' = W$. By rule (S-VAR) we have $\Gamma \vdash X \leq U : \kappa$, by assumption we have $\Gamma \vdash X\vec{V} : *$ and by inversion of kinding $\Gamma \vdash X : \vec{\kappa} \rightarrow *$ and $\Gamma \vdash V_i : \kappa_i$ for all $i = 1 \dots |\vec{\kappa}| =: n$ and by induction hypothesis we have $\Gamma \vdash U @^{\vec{\kappa}} \vec{V} \leq W : *$. So we can derive:

$$\frac{\frac{\Gamma \vdash X \leq U : \kappa \quad (\Gamma \vdash V_i : \kappa_i)_{i=1 \dots n}}{\Gamma \vdash X\vec{V} \leq U\vec{V} : *} \quad \frac{\frac{\Gamma \vdash U\vec{V} = U @^{\vec{\kappa}} \vec{V} : *}{\Gamma \vdash U\vec{V} \leq U @^{\vec{\kappa}} \vec{V} : *} \text{ (S-EQ)} \quad \Gamma \vdash U @^{\vec{\kappa}} \vec{V} \leq W : *}{\Gamma \vdash U\vec{V} \leq W : *} \text{ (S-TRANS)}}{\Gamma \vdash X\vec{V} \leq W : *} \text{ (S-TRANS)}$$

Case (SA-ABS). We have $V = \lambda \vec{X}.W$ and $V' = \lambda \vec{X}.W'$. We have by induction hypothesis $\Gamma, (X_i \leq \top_{\kappa_i} : \kappa_i)_i \vdash W \leq W' : \vec{\kappa} \rightarrow *$. So, we finish with the rule (S-ABS-VECT)

$$\frac{\Gamma, (X_i \leq \top_{\kappa_i} : \kappa_i)_i \vdash W \leq W' : \vec{\kappa} \rightarrow *}{\Gamma \vdash \lambda \vec{X}.W \leq \lambda \vec{X}.W' : *}$$
 (S-ABS-VECT)

Case (SA-ARROW). We have $V = U \rightarrow U'$ and $V' = W \rightarrow W'$. By induction hypothesis we have $\Gamma \vdash W \leq U : *$ and $\Gamma \vdash U' \leq W' : *$. So we can apply (S-ARROW).

Case (SA-ALL). We have $V = \forall X \leq U : \kappa.W$ and $V' = \forall X \leq U : \kappa.W'$. By induction hypothesis we have $\Gamma, X \leq U : \kappa \vdash W \leq W' : *$. So we can apply (S-ALL). \square

Lemma 3.1.2 *Let $\Gamma \vdash F, F' : \kappa$. If $\mathcal{D} :: \text{nf}(\Gamma) \vdash \text{nf}(F) \leq \text{nf}(F') : \kappa$, then $\text{nf}(\Gamma) \vdash F \leq F' : \kappa$.*

Proof. Let Γ' be $\text{nf}(\Gamma)$. We have $\Gamma' \vdash F = \text{nf}(F) : \kappa$ and $\Gamma' \vdash F' = \text{nf}(F') : \kappa$ by lemmas 2.1.4 and 2.3.2 so we can derive:

$$\frac{\frac{\Gamma' \vdash F = \text{nf}(F) : \kappa}{\Gamma' \vdash F \leq \text{nf}(F) : \kappa} \text{ (S-EQ)} \quad \Gamma' \vdash \text{nf}(F) \leq \text{nf}(F') : \kappa}{\Gamma' \vdash F \leq \text{nf}(F') : \kappa} \text{ (S-TRANS)} \quad \frac{\Gamma' \vdash F' = \text{nf}(F') : \kappa}{\Gamma' \vdash \text{nf}(F') = F' : \kappa} \quad \frac{\Gamma' \vdash \text{nf}(F') \leq F' : \kappa}{\Gamma' \vdash F \leq F' : \kappa}$$

\square

Theorem 3.1.3 (Soundness of algorithmic subtyping) *Let $\Gamma \vdash F, F' : \kappa$. If $\mathcal{D} :: \text{nf}(\Gamma) \vdash_a \text{nf}(F) \leq \text{nf}(F') : \kappa$, then $\Gamma \vdash F \leq F' : \kappa$.*

Proof.

We have $\text{nf}(\Gamma) \vdash_a \text{nf}(F) \leq \text{nf}(F') : \kappa$, so, by lemma 3.1.1 we have $\text{nf}(\Gamma) \vdash \text{nf}(F) \leq \text{nf}(F') : \kappa$, and by lemma 3.1.2 we have $\text{nf}(\Gamma) \vdash F \leq F' : \kappa$, and by lemma 2.1.4 $\Gamma \vdash F \leq F' : \kappa$. \square

3.2. Completeness of algorithmic subtyping

Completeness of the algorithm means that any derivable statement in the declarative system is also derivable in the syntax-directed system. This is more difficult to show than soundness, because we have eliminated declarative rules like (S-TRANS) or (S-APP). Thus we need to show that these rules are still admissible in the algorithmic system.

Lemma 3.2.1 (Abstraction) $\mathcal{D}_1 :: \Gamma, X \leq \top_{\kappa} : \kappa \vdash_a V \leq V' : \kappa'$ iff $\mathcal{D}_2 :: \Gamma \vdash_a \lambda X V \leq \lambda X V' : \kappa \rightarrow \kappa'$.

Proof.

“ \Rightarrow ”

In case of $\kappa' = *$ the proof can be completed with (SA-Abs). Otherwise let $\kappa' = \vec{\kappa} \rightarrow *$ with $\vec{\kappa}$ nonempty.

The last rule of \mathcal{D}_1 must have been (SA-Abs) with $V = \lambda\vec{X}U$ and $V' = \lambda\vec{X}U'$.

$$(1) \quad \frac{\Gamma, X \leq \top_{\kappa} : \kappa, (X_i \leq \top_{\kappa_i}) : \kappa_i \vdash_{\mathbf{a}} U \leq U' : *}{\Gamma, X \leq \top_{\kappa} : \kappa \vdash_{\mathbf{a}} \lambda\vec{X}.U \leq \lambda\vec{X}.U' : \vec{\kappa} \rightarrow *} \quad (\text{SA-Abs})$$

Then we apply (SA-Abs) to (1) and we get:

$$(1) \quad \frac{\Gamma, X \leq \top_{\kappa} : \kappa, X_i \leq \top_{\kappa_i} : \kappa_i \vdash_{\mathbf{a}} U \leq U' : *}{\Gamma \vdash_{\mathbf{a}} \lambda X. \lambda \vec{X}. U \leq \lambda X. \lambda \vec{X}. U' : \kappa \rightarrow \vec{\kappa} \rightarrow *} \quad (\text{SA-Abs})$$

$$\iff \Gamma \vdash_{\mathbf{a}} \lambda X. V \leq \lambda X. V' : \kappa \rightarrow \kappa'$$

“ \Leftarrow ”

In case of $\kappa' = *$ the proof can be completed with inversion. Otherwise let $\kappa' = \vec{\kappa} \rightarrow *$.

The last rule of \mathcal{D}_2 must have been (SA-Abs) with $V = \lambda\vec{X}U$ and $V' = \lambda\vec{X}U'$.

$$(2) \quad \frac{\Gamma, X \leq \top_{\kappa}, X_i \leq \top_{\kappa_i} \vdash_{\mathbf{a}} U \leq U' : *}{\Gamma \vdash_{\mathbf{a}} \lambda X \lambda \vec{X} U \leq \lambda X \lambda \vec{X} U' : \kappa \rightarrow \vec{\kappa} \rightarrow *} \quad (\text{SA-Abs})$$

Then we can apply (SA-Abs) to (2) and we are finished:

$$(2) \quad \frac{\Gamma, X \leq \top_{\kappa}, X_i \leq \top_{\kappa_i} \vdash_{\mathbf{a}} U \leq U' : *}{\Gamma, X \leq \top_{\kappa} \vdash_{\mathbf{a}} \lambda \vec{X} U \leq \lambda \vec{X} U' : \vec{\kappa} \rightarrow *} \quad (\text{SA-Abs})$$

$$\iff \Gamma, X \leq \top_{\kappa} \vdash_{\mathbf{a}} V \leq V' : \kappa'$$

□

Lemma 3.2.2 (Reflexivity) *If V is β -normal then $\Gamma \vdash_{\mathbf{a}} V \leq V : \kappa$.*

Proof. By induction on V .

Case $V \rightarrow V'$

By induction hypothesis,
$$\frac{\Gamma \vdash_{\mathbf{a}} V \leq V : \kappa \quad \Gamma \vdash_{\mathbf{a}} V' \leq V' : \kappa}{\Gamma \vdash_{\mathbf{a}} V \rightarrow V' \leq V \rightarrow V' : \kappa} \quad (\text{SA-ARROW})$$

Case λXV

By induction hypothesis,
$$\frac{\Gamma, X \leq \top_{\kappa} : \kappa \vdash_{\mathbf{a}} V \leq V : \kappa'}{\Gamma \vdash_{\mathbf{a}} \lambda XV \leq \lambda XV : \kappa \rightarrow \kappa'}$$
 by Lemma 3.2.1 (Abstraction)

Case $\forall X \leq V : \kappa. W$

By induction hypothesis,
$$\frac{\Gamma, X \leq V : \kappa \vdash_{\mathbf{a}} W \leq W : *}{\Gamma \vdash_{\mathbf{a}} \forall X \leq V : \kappa. W \leq \forall X \leq V : \kappa. W : *} \quad (\text{SA-ALL})$$

Case \top

We have immediately: $\frac{}{\Gamma \vdash_{\mathbf{a}} \top \leq \top : *}$ (SA-TOP)

Case $X\vec{V}$

Follows with (SA-REFL): $\frac{}{\Gamma \vdash_{\mathbf{a}} X\vec{V} \leq X\vec{V} : *}$ (SA-REFL)

□

Lemma 3.2.3 (Transitivity) $\mathcal{D}_1 :: \Gamma \vdash_{\mathbf{a}} V_1 \leq V_2 : \kappa$ and $\mathcal{D}_2 :: \Gamma \vdash_{\mathbf{a}} V_2 \leq V_3 : \kappa$ imply $\Gamma \vdash_{\mathbf{a}} V_1 \leq V_3 : \kappa$.

Proof. By induction on \mathcal{D}_1 .

Case

$$\Gamma \vdash_{\mathbf{a}} V_1 \leq V_2 : * \quad \frac{}{\Gamma \vdash_{\mathbf{a}} V_2 \leq \top : *} \text{ (SA-TOP)}$$

Follows with (SA-TOP): $\frac{}{\Gamma \vdash_{\mathbf{a}} V_1 \leq \top : *} \text{ (SA-TOP)}$

Case

$$\frac{}{\Gamma \vdash_{\mathbf{a}} V_1 \leq \top : *} \text{ (SA-TOP)} \quad \Gamma \vdash_{\mathbf{a}} \top \leq V_3 : *$$

By looking at the rules we see that V_3 must be \top and \mathcal{D}_2 must end by (SA-TOP).

Case

$$\frac{}{\Gamma \vdash_{\mathbf{a}} X\vec{V} \leq X\vec{V} : *} \text{ (SA-REFL)} \quad \Gamma \vdash_{\mathbf{a}} X\vec{V} \leq V_3 : *$$

\mathcal{D}_2 is the desired result.

Case

$$\frac{\Gamma \vdash_{\mathbf{a}} V_2 \leq V_1 : * \quad \Gamma \vdash_{\mathbf{a}} V'_1 \leq V'_2 : *}{\Gamma \vdash_{\mathbf{a}} V_1 \rightarrow V'_1 \leq V_2 \rightarrow V'_2 : *} \text{ (SA-ARROW)}$$

$$\frac{\Gamma \vdash_{\mathbf{a}} V_3 \leq V_2 : * \quad \Gamma \vdash_{\mathbf{a}} V'_2 \leq V'_3 : *}{\Gamma \vdash_{\mathbf{a}} V_2 \rightarrow V'_2 \leq V_3 \rightarrow V'_3 : *} \text{ (SA-ARROW)}$$

Using the induction hypothesis, we conclude:

$$\frac{\Gamma \vdash_{\mathbf{a}} V_3 \leq V_1 : * \quad \Gamma \vdash_{\mathbf{a}} V'_1 \leq V'_3 : *}{\Gamma \vdash_{\mathbf{a}} V_1 \rightarrow V'_1 \leq V_3 \rightarrow V'_3 : *} \text{ (SA-ARROW)}$$

Case

$$\frac{\Gamma, X \leq V : \kappa \vdash_{\mathbf{a}} W_1 \leq W_2 : *}{\Gamma \vdash_{\mathbf{a}} \forall X \leq V : \kappa. W_1 \leq \forall X \leq V : \kappa. W_2 : *} \text{ (SA-ALL)}$$

$$\frac{\Gamma, X \leq V : \kappa \vdash_{\mathbf{a}} W_2 \leq W_3 : *}{\Gamma \vdash_{\mathbf{a}} \forall X \leq V : \kappa. W_2 \leq \forall X \leq V : \kappa. W_3 : *} \text{ (SA-ALL)}$$

Using the induction hypothesis, we conclude:

$$\frac{\Gamma, X \leq V : \kappa \vdash_{\mathbf{a}} W_1 \leq W_3 : *}{\Gamma \vdash_{\mathbf{a}} \forall X \leq V : \kappa. W_1 \leq \forall X \leq V : \kappa. W_3 : *} \text{ (SA-ALL)}$$

Case

$$\frac{\Gamma, (X_i \leq \top_{\kappa_i} : \kappa_i) \vdash_a V_1 \leq V_2 : *}{\Gamma \vdash_a \lambda \vec{X}. V_1 \leq \lambda \vec{X}. V_2 : \vec{\kappa} \rightarrow *} \text{ (SA-ABS)}$$

$$\frac{\Gamma, (X_i \leq \top_{\kappa_i} : \kappa_i) \vdash_a V_2 \leq V_3 : *}{\Gamma \vdash_a \lambda \vec{X}. V_2 \leq \lambda \vec{X}. V_3 : \vec{\kappa} \rightarrow *} \text{ (SA-ABS)}$$

Using the induction hypothesis, we conclude:

$$\frac{\Gamma, (X_i \leq \top_{\kappa_i} : \kappa_i) \vdash_a V_1 \leq V_3 : *}{\Gamma \vdash_a \lambda \vec{X}. V_1 \leq \lambda \vec{X}. V_3 : \vec{\kappa} \rightarrow *} \text{ (SA-ABS)}$$

Case

$$\frac{\Gamma \vdash_a U @^{\vec{\kappa}} \vec{V} \leq V_2 : * \quad (X \leq U : \kappa) \in \Gamma}{\Gamma \vdash_a X \vec{V} \leq V_2 : *} \text{ (SA-BOUND)} \quad \Gamma \vdash_a V_2 \leq V_3 : *$$

Using the induction hypothesis, we conclude:

$$\frac{\Gamma \vdash_a U @^{\vec{\kappa}} \vec{V} \leq V_3 : * \quad (X \leq U : \kappa) \in \Gamma}{\Gamma \vdash_a X \vec{V} \leq V_3 : *} \text{ (SA-BOUND)}$$

□

The following substitution lemma is the key step in showing that algorithmic subtyping is closed under application, i.e., complete for rule (S-APP).

Lemma 3.2.4 (Substitution) *Let $\Gamma, X : \kappa, \Gamma' \vdash V, V' \uparrow \kappa'$ and $\Gamma \vdash U \uparrow \kappa$. If $\mathcal{D} :: \Gamma, X : \kappa, \Gamma' \vdash_a V \leq V' : \kappa'$ then $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X]V \leq [U^\kappa/X]V' : \kappa'$.*

Proof. By induction on \mathcal{D} .

Case (SA-REFL) We have $\Gamma, X : \kappa, \Gamma' \vdash_a Y \vec{V} \leq Y \vec{V} : *$ with Y possibly identical to X and we show: $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X](Y \vec{V}) \leq [U^\kappa/X](Y \vec{V}) : *$. This follows by the reflexivity lemma (Lemma 3.2.2).

Case (SA-TOP) We have $\Gamma, X : \kappa, \Gamma' \vdash_a V \leq \top : *$ and we show: $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X]V \leq [U^\kappa/X]\top : *$, i.e., $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X]V \leq \top : *$. We can finish with (SA-TOP).

Case (SA-ARROW) We have $\Gamma, X : \kappa, \Gamma' \vdash_a V \rightarrow V' \leq W \rightarrow W' : *$ and we show: $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X](V \rightarrow V') \leq [U^\kappa/X](W \rightarrow W') : *$, i.e., $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X]V \rightarrow [U^\kappa/X]V' \leq [U^\kappa/X]W \rightarrow [U^\kappa/X]W' : *$.

We have by induction hypothesis $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X]W \leq [U^\kappa/X]V : *$ and $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X]V' \leq [U^\kappa/X]W' : *$. Thus we can finish with (SA-ARROW).

Case (SA-ALL) We have $\Gamma, X : \kappa, \Gamma' \vdash_a \forall Y \leq V : \kappa. W \leq \forall Y \leq V : \kappa. W' : *$ and we show $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X](\forall Y \leq V : \kappa. W) \leq [U^\kappa/X](\forall Y \leq V : \kappa. W') : *$, i.e.,

$$\Gamma, [U^\kappa/X]\Gamma' \vdash_a \forall Y \leq [U^\kappa/X]V : \kappa. [U^\kappa/X]W \leq \forall Y \leq [U^\kappa/X]V : \kappa. [U^\kappa/X]W' : *$$

By induction hypothesis we have, $\Gamma, [U^\kappa/X]\Gamma', Y \leq [U^\kappa/X]V : \kappa \vdash_a [U^\kappa/X]W \leq [U^\kappa/X]W' : *$ thus we can finish with (SA-ALL).

Case (SA-ABS) We have $\Gamma, X : \kappa, \Gamma' \vdash_a \lambda \vec{X}.V \leq \lambda \vec{X}.V' : \vec{\kappa} \rightarrow *$ and we show: $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X](\lambda \vec{X}.V) \leq [U^\kappa/X](\lambda \vec{X}.V') : \vec{\kappa} \rightarrow *$, i.e.,

$$\Gamma, [U^\kappa/X]\Gamma' \vdash_a \lambda \vec{X}.[U^\kappa/X]V \leq \lambda \vec{X}.[U^\kappa/X]V' : \vec{\kappa} \rightarrow *$$

By induction hypothesis we have, $\Gamma, [U^\kappa/X]\Gamma', X_i : \kappa_i \vdash_a [U^\kappa/X]V \leq [U^\kappa/X]V' : *$ thus we can finish with (SA-ABS).

Case (SA-BOUND) We have $\Gamma, X : \kappa, \Gamma' \vdash_a Y\vec{V} \leq W : *$ and there is an U' with $Y \leq U' \in \Gamma, X : \kappa, \Gamma'$ and we show: $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X](Y\vec{V}) \leq [U^\kappa/X]W : *$,

Subcase $X = Y$ Then we have $\Gamma, X \leq \top_\kappa : \kappa, \Gamma' \vdash_a X\vec{V} \leq W : *$. Then we have by assumption $\Gamma, X \leq \top_\kappa : \kappa, \Gamma' \vdash_a \top_\kappa @^{\vec{\kappa}} \vec{V} \leq W : *$ i.e., $\Gamma, X \leq \top_\kappa : \kappa, \Gamma' \vdash_a \top \leq W : *$ thus, we get by inversion: $W = \top$, and we show $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X](X\vec{V}) \leq \top : *$ and we finish with (SA-TOP).

Subcase $X \neq Y$ Then we have $\Gamma, X : \kappa, \Gamma' \vdash_a Y\vec{V} \leq W : *$ and there is an U' with $Y \leq U' \in \Gamma, X : \kappa, \Gamma'$ and we show: $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X](Y\vec{V}) \leq [U^\kappa/X]W : *$. W.l.o.g. $Y \notin FV(U)$, thus, our goal becomes $\Gamma, [U^\kappa/X]\Gamma' \vdash_a Y[U^\kappa/X]\vec{V} \leq [U^\kappa/X]W : *$.

By induction hypothesis we have $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X](U' @^{\vec{\kappa}} \vec{V}) \leq [U^\kappa/X]W : *$, which is equivalent to $\Gamma, [U^\kappa/X]\Gamma' \vdash_a [U^\kappa/X]U' @^{\vec{\kappa}} [U^\kappa/X]\vec{V} \leq [U^\kappa/X]W : *$ by lemma 2.2.4, part (1), thus, we can finish with (SA-BOUND). \square

Completeness can be shown now by induction on derivations, using the previous lemmas: abstraction, reflexivity, transitivity and substitution.

Theorem 3.2.5 (Completeness) *If $\mathcal{D} :: \Gamma \vdash F \leq F' : \kappa$. then $\text{nf}(\Gamma) \vdash_a \text{nf}(F) \leq \text{nf}(F') : \kappa$.*

Proof. By induction on \mathcal{D} .

Case (S-VAR)

$$\frac{\Gamma \vdash (X \leq G : \kappa) \in \Gamma}{\Gamma \vdash X \leq G : \kappa} \quad (\text{S-VAR})$$

We show: $\text{nf}(\Gamma) \vdash_a \text{nf}(X) \leq \text{nf}(G) : \kappa$. We know:

$$\begin{aligned} \text{nf}(G) &= \lambda \vec{Y}.V && : \vec{\kappa} \rightarrow * \text{ and} \\ \text{nf}(X) &= \lambda \vec{Y}.X \text{nf}(\vec{Y}) && : \vec{\kappa} \rightarrow * \end{aligned}$$

Thus we show: $\text{nf}(\Gamma) \vdash_a \lambda \vec{Y}.X \text{nf}(\vec{Y}) \leq \lambda \vec{Y}.V : \kappa$. Note that if $X \leq G : \kappa \in \Gamma$ then $X \leq \lambda \vec{Y}.V : \vec{\kappa} \rightarrow * \in \text{nf}(\Gamma)$.

Moreover we have $[(\text{nf}(\vec{Y}))^{\vec{\kappa}}/\vec{Y}]V \equiv V$ by lemma 2.4.5. Hence the following follows from the reflexivity lemma (Lemma 3.2.2): $\text{nf}(\Gamma), \vec{Y} \leq \top_{\vec{\kappa}} : \vec{\kappa} \vdash (\lambda \vec{Y}.V) @^{\vec{\kappa}} \text{nf}(\vec{Y}) \leq V : *$. So we can derive:

$$\frac{\frac{(X \leq \lambda \vec{Y}.V : \vec{\kappa} \rightarrow *) \in \text{nf}(\Gamma) \quad \text{nf}(\Gamma), \vec{Y} \leq \top_{\vec{\kappa}} : \vec{\kappa} \vdash_a \overbrace{[(\text{nf}(\vec{Y}))^{\vec{\kappa}}/\vec{Y}]V}^V \leq V : *}{\text{nf}(\Gamma), \vec{Y} \leq \top_{\vec{\kappa}} : \vec{\kappa} \vdash_a X \text{nf}(\vec{Y}) \leq V : *} \quad (\text{SA-ABS})}{\text{nf}(\Gamma) \vdash_a \lambda \vec{Y}.X \text{nf}(\vec{Y}) \leq \lambda \vec{Y}.V : \kappa} \quad (\text{SA-BOUND})$$

Case (S-APP)

$$\frac{\Gamma \vdash F \leq G : \kappa \rightarrow \kappa' \quad \Gamma \vdash H : \kappa}{\Gamma \vdash FH \leq GH : \kappa'} \quad (\text{S-APP})$$

We show $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(FH) \leq \text{nf}(GH) : \kappa'$

Let $\text{nf}(F) = \lambda X.V : \kappa \rightarrow \kappa'$, $\text{nf}(G) = \lambda X.V' : \kappa \rightarrow \kappa'$ and $\text{nf}(H) = U$.

$$\begin{aligned} \text{nf}(FH) &= \text{nf}((\lambda X.V)U) = [U^\kappa/X]V \\ \text{nf}(GH) &= \text{nf}((\lambda X.V')U) = [U^\kappa/X]V' \end{aligned}$$

By induction hypothesis,

$$\begin{aligned} \text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(F) \leq \text{nf}(G) : \kappa \rightarrow \kappa' &\iff \\ \text{nf}(\Gamma) \vdash_{\mathbf{a}} \lambda X.V \leq \lambda X.V' : \kappa \rightarrow \kappa' & \end{aligned}$$

With Lemma 3.2.1 (Abstraction) we get:

$$\text{nf}(\Gamma), X \leq \top_\kappa : \kappa \vdash_{\mathbf{a}} V \leq V' : \kappa'$$

Then we have to show: $\text{nf}(\Gamma) \vdash_{\mathbf{a}} [U^\kappa/X]V \leq [U^\kappa/X]V' : \kappa'$.

This follows by the substitution lemma (Lemma 3.2.4).

Case (S-ABS)

$$\frac{\Gamma, X \leq \top_\kappa \vdash F \leq F' : \kappa'}{\Gamma \vdash \lambda X.F \leq \lambda X.F' : \kappa \rightarrow \kappa'} \quad (\text{S-ABS})$$

We show $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(\lambda X.F) \leq \text{nf}(\lambda X.F') : \kappa \rightarrow \kappa'$.

By induction hypothesis, $\text{nf}(\Gamma), X \leq \top_\kappa \vdash_{\mathbf{a}} \text{nf}(F) \leq \text{nf}(F') : \kappa'$. The goal follows by the abstraction lemma (Lemma 3.2.1).

Case (S-ARROW)

$$\frac{\Gamma \vdash B_1 \leq A_1 : * \quad \Gamma \vdash A_2 \leq B_2 : *}{\Gamma \vdash A_1 \rightarrow A_2 \leq B_1 \rightarrow B_2 : *} \quad (\text{S-ARROW})$$

We show: $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(A_1 \rightarrow A_2) \leq \text{nf}(B_1 \rightarrow B_2) : *$, i.e., $\text{nf}(\Gamma) \vdash \text{nf}(A_1) \rightarrow \text{nf}(A_2) \leq \text{nf}(B_1) \rightarrow \text{nf}(B_2) : *$. By induction hypothesis, $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(B_1) \leq \text{nf}(A_1) : *$ and $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(A_2) \leq \text{nf}(B_2) : *$, thus, we can finish with (SA-ARROW).

Case (S-ALL)

$$\frac{\Gamma, X \leq G : \kappa \vdash A \leq B : *}{\Gamma \vdash \forall X \leq G : \kappa. A \leq \forall X \leq G : \kappa. B : *} \quad (\text{S-ALL})$$

We show: $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(\forall X \leq G : \kappa. A) \leq \text{nf}(\forall X \leq G : \kappa. B) : *$, i.e.,

$\text{nf}(\Gamma) \vdash_{\mathbf{a}} (\forall X \leq \text{nf}(G) : \kappa. \text{nf}(A)) \leq (\forall X \leq \text{nf}(G) : \kappa. \text{nf}(B)) : *$. By induction hypothesis, $\text{nf}(\Gamma), X \leq \text{nf}(G) : \kappa \vdash_{\mathbf{a}} \text{nf}(A) \leq \text{nf}(B) : *$ thus we can finish with (SA-ALL).

Case (S-Top)

$$\frac{\Gamma \vdash A : *}{\Gamma \vdash A \leq \top : *} \text{ (S-Top)}$$

We show: $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(A) \leq \text{nf}(\top) : *$, i.e., $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(A) \leq \top : *$. We can finish with (SA-Top).

Case (S-EQ)

$$\frac{\Gamma \vdash F = G : \kappa}{\Gamma \vdash F \leq G : \kappa} \text{ (S-EQ)}$$

We show: $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(F) \leq \text{nf}(G) : \kappa$

$\Gamma \vdash F = G : \kappa$ implies $\text{nf}(F) \equiv \text{nf}(G)$ by theorem 2.5.2. We can then apply the reflexivity lemma (Lemma 3.2.2) and we get: $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(F) \leq \text{nf}(G) : \kappa$.

Case

$$\frac{\Gamma \vdash F \leq G : \kappa \quad \Gamma \vdash G \leq H : \kappa}{\Gamma \vdash F \leq H : \kappa} \text{ (S-TRANS)}$$

We show: $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(F) \leq \text{nf}(H) : \kappa$

By induction hypothesis, $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(F) \leq \text{nf}(G) : \kappa$ and $\text{nf}(\Gamma) \vdash_{\mathbf{a}} \text{nf}(G) \leq \text{nf}(H) : \kappa$. The goal follows with the transitivity lemma (Lemma 3.2.3). □

3.3. Termination of algorithmic subtyping

We now show that the algorithm we developed is indeed a decision procedure for the subtype relation. We must verify that this recursively defined procedure is really an algorithm - that it halts in finite time on all well-kinded inputs.

Compagnoni and Goguen [CG03] leave the termination of algorithmic subtyping for $F_{<}^{\omega}$ open. However, it is not trivial, problematic is the rule for upper bounds.

$$\frac{(X \leq U : \kappa) \in \Gamma \quad \Gamma \vdash_{\mathbf{a}} U @^{\vec{\kappa}} \vec{V} \leq W : *}{\Gamma \vdash_{\mathbf{a}} X \vec{V} \leq W : *}$$

Define $\kappa_0 := *$ and $\kappa_{n+1} := \kappa_n \rightarrow \kappa_n$. Let $2 := \lambda X \lambda Y. X (X Y)$. Let $\Gamma := (X_i \leq 2 : \kappa_{i+1})_{i=1..n}$. Then consider the problem

$$\Gamma \vdash_{\mathbf{a}} X_n X_{n-1} \dots X_1 (\lambda Z. Z) \leq \lambda Z. Z : \kappa_1$$

There is no obvious termination measure. Application of the bound rule will not always remove one variable from the l.h.s. But it may create new β -redexes, whose reduction leads to duplication of variables. I conjecture the number of applications of the bound rule is something like 2_n which is a tower of 2s of height n .

Pierce and Steffen [PS97] prove termination by showing strong normalization of $\beta\top\Gamma$ -reduction. The $\beta\top\Gamma$ -reduction is an extension of the usual β -reduction, where variables are allowed to be replaced by their upper bounds from the context.

The trick to show s.n. of Γ -reduction is to prepare for all bound-lookups in the kinding derivation. Their rule is

$$\frac{(X \leq G : \kappa) \in \Gamma \quad \Gamma \vdash G : \kappa}{\Gamma \vdash X : \kappa}$$

Normally, one would only require the context Γ to be well-formed to conclude $\Gamma \vdash X : \kappa$. Here, they embed a derivation of the well-kindedness of G into the well-kindedness of X . Hence, when performing a Γ -reduction $X \rightarrow G$, the well-kindedness derivation gets smaller, ensuring termination of Γ -reduction. We aim at using this idea for a more direct proof of termination of algorithmic subtyping. The proof is by induction on derivations. It is straightforward, because the judgement $\Gamma \vdash V \uparrow \kappa$ was designed for this purpose (s. Chapter 2).

Theorem 3.3.1 (Termination of algorithmic subtyping) *If $\mathcal{D}_1 :: \Gamma \vdash V \uparrow \kappa$ and $\mathcal{D}_2 :: \Gamma \vdash V' \uparrow \kappa$ then the query $\Gamma \vdash_a V \leq V' : \kappa$ terminates.*

Proof. By simultaneous induction on $\{\mathcal{D}_1, \mathcal{D}_2\}$.

$$\text{Case } \mathcal{D}_1 :: \Gamma \vdash V \uparrow \kappa \quad \mathcal{D}_2 :: \frac{\Gamma \uparrow}{\Gamma \vdash \top \uparrow *} \text{ (LN-TOP)}$$

We show $\Gamma \vdash_a V \leq \top : *$ terminates.

We can apply $\frac{}{\Gamma \vdash_a V \leq \top : *}$ (SA-TOP) and we are done.

$$\text{Case } \mathcal{D}_1 :: \frac{\Gamma \uparrow}{\Gamma \vdash \top \uparrow *} \text{ (LN-TOP)} \quad \mathcal{D}_2 :: \Gamma \vdash V' \uparrow *$$

Subcase $V' = \top$, follows from the first case.

Subcase $V' \neq \top$, the query $\Gamma \vdash \top \leq V' : *$ fails, and terminates.

$$\text{Case } \mathcal{D}_1 :: \frac{\Gamma, X : \kappa \vdash W \uparrow \kappa'}{\Gamma \vdash \lambda X W \uparrow \kappa \rightarrow \kappa'} \text{ (LN-ABS)} \quad \mathcal{D}_2 :: \Gamma \vdash V' \uparrow \kappa \rightarrow \kappa'.$$

We show: $\Gamma \vdash \lambda X W \leq V' : \kappa \rightarrow \kappa'$ terminates.

$$\text{Then } V' = \lambda X W', \quad \mathcal{D}_2 :: \frac{\Gamma, X : \kappa \vdash W' \uparrow \kappa'}{\Gamma \vdash \lambda X W' \uparrow \kappa \rightarrow \kappa'} \text{ (LN-ABS)}$$

$$(1) \quad \frac{\Gamma, X \leq \top : \kappa \vdash_a W \leq W' : \kappa \rightarrow \kappa'}{\Gamma \vdash_a \lambda X W \leq \lambda X W' : \kappa \rightarrow \kappa'} \text{ Lemma Abstraction (Lemma 3.2.1)}$$

By induction hypothesis the query (1) terminates, and so does the desired query.

$$\text{Case } \mathcal{D}_1 :: \frac{\Gamma \vdash V_1 \uparrow * \quad \Gamma \vdash W_1 \uparrow *}{\Gamma \vdash V_1 \rightarrow W_1 \uparrow *} \text{ (LN-ARROW)} \quad \mathcal{D}_2 :: \Gamma \vdash V' \uparrow *.$$

We show: $\Gamma \vdash V_1 \rightarrow W_1 \leq V' : *$ terminates.

$$\text{Subcase } V' = V_2 \rightarrow W_2 \uparrow *, \quad \mathcal{D}_2 :: \frac{\Gamma \vdash V_2 \uparrow * \quad \Gamma \vdash W_2 \uparrow *}{\Gamma \vdash V_2 \rightarrow W_2 \uparrow *} \text{ (LN-ARROW)}$$

Then we have

$$\frac{(1) \quad \Gamma \vdash_a V_2 \leq V_1 : * \quad (2) \quad \Gamma \vdash_a W_1 \leq W_2 : *}{\Gamma \vdash_a V_1 \rightarrow W_1 \leq V_2 \rightarrow W_2 : *} \text{ (SA-ARROW)}$$

By induction hypothesis the queries (1) and (2) terminate, and so does the desired query as well.

Subcase $V' \neq V_2 \rightarrow W_2 \uparrow *$ then the query $\Gamma \vdash_a V_1 \rightarrow W_1 \leq V' : *$ fails, and terminates.

$$\text{Case } \mathcal{D}_1 :: \frac{\Gamma, X \leq U : \kappa \vdash W \uparrow *}{\Gamma \vdash \forall X \leq U : \kappa. W \uparrow *} \text{ (LN-ALL)} \quad \mathcal{D}_2 :: \Gamma \vdash V' \uparrow *$$

We show: $\Gamma \vdash \forall X \leq U : \kappa. W \leq V' : *$ terminates.

$$\text{Subcase } V' = \forall X \leq U : \kappa. W', \quad \mathcal{D}_2 :: \frac{\Gamma, X \leq U : \kappa \vdash W' \uparrow *}{\Gamma \vdash \forall X \leq U : \kappa. W' \uparrow *} \text{ (LN-ALL)} \text{ Then we have}$$

$$\frac{(1) \quad \Gamma, X \leq U \vdash_a W \leq W' : *}{\Gamma \vdash_a \forall X \leq U. W \leq \forall X \leq U. W' : *} \text{ (SA-ALL)}$$

By induction hypothesis the query (1) terminates, and so does the desired query as well.

Subcase $V' \neq \forall X \leq U : \kappa. W'$ then the query $\Gamma \vdash_a \forall X \leq U : \kappa. W \leq V' : *$ fails, and terminates.

$$\text{Case } \mathcal{D}_1 :: \frac{(X \leq U : \vec{\kappa} \rightarrow *) \in \Gamma \quad \Gamma \vdash V_i \uparrow \kappa_i \text{ for all } i \quad \Gamma \vdash U @^{\vec{\kappa}} \vec{V} \uparrow *}{\Gamma \vdash X \vec{V} \uparrow *} \text{ (LN-VAR-BOUND)}$$

$$\mathcal{D}_2 :: \Gamma \vdash V' \uparrow *$$

We show: $\Gamma \vdash_a X \vec{V} \leq V' : *$ terminates.

Subcase $V' = X \vec{V}$,

then we have $\frac{}{\Gamma \vdash_a X \vec{V} \leq X \vec{V}} \text{ (SA-REFL)}$ and the query terminates immediately.

Subcase $V' \neq X \vec{V}$

$$\text{We have } \frac{(X \leq U : \kappa) \in \Gamma \quad \Gamma \vdash_a U @^{\vec{\kappa}} \vec{V} \leq V' : *}{\Gamma \vdash_a X \vec{V} \leq V' : *} \text{ (SA-BOUND)}$$

By the induction hypothesis, $\Gamma \vdash_a U @^{\vec{\kappa}} \vec{V} \leq V' : *$ terminates, and so does the desired query as well.

□

3.4. Example

Let List be defined by

$$\text{List} = \lambda X.\forall R:*. (A \rightarrow R \rightarrow R) \rightarrow R \rightarrow R$$

as in [Pie02, Chapter 23.4]. Now we can use lists to test our algorithm. We can derive the following subtyping rule: $\frac{\Gamma, R:*\vdash A \leq B:*\quad}{\Gamma \vdash \text{List } A \leq \text{List } B:*$

Note that the types are not in normal form. So we have to calculate their long normal form in order to apply the algorithmic rules.

$$\begin{aligned} \text{List } A &= (\lambda X.\forall R:*. (X \rightarrow R \rightarrow R) \rightarrow R \rightarrow R)A \\ &=_{\beta} \forall R:*. (A \rightarrow R \rightarrow R) \rightarrow R \rightarrow R \\ \text{List } B &= (\lambda X.\forall R:*. (X \rightarrow R \rightarrow R) \rightarrow R \rightarrow R)B \\ &=_{\beta} \forall R:*. (B \rightarrow R \rightarrow R) \rightarrow R \rightarrow R \end{aligned}$$

Moreover $\vdash \forall R:*. (A \rightarrow R \rightarrow R) \rightarrow R \rightarrow R \uparrow *$ and $\vdash \forall R:*. (B \rightarrow R \rightarrow R) \rightarrow R \rightarrow R \uparrow *$ by the rule (LN-ALL). Thus we can derive:

$$\frac{\frac{R:*\vdash A \leq B \quad \frac{\frac{R:*\vdash R \leq R \quad R:*\vdash R \leq R}{R:*\vdash R \rightarrow R \leq R \rightarrow R}}{R:*\vdash B \rightarrow R \rightarrow R \leq A \rightarrow R \rightarrow R} \quad \frac{\frac{R:*\vdash R \leq R \quad R:*\vdash R \leq R}{R:*\vdash R \rightarrow R \leq R \rightarrow R}}{R:*\vdash (A \rightarrow R \rightarrow R) \rightarrow R \rightarrow R \leq (B \rightarrow R \rightarrow R) \rightarrow R \rightarrow R} \text{ (SA-REFL)}}{\frac{R:*\vdash (A \rightarrow R \rightarrow R) \rightarrow R \rightarrow R \leq (B \rightarrow R \rightarrow R) \rightarrow R \rightarrow R}{\forall R \leq \top:*. (A \rightarrow R \rightarrow R) \rightarrow R \rightarrow R \leq \forall R \leq \top:*. (B \rightarrow R \rightarrow R) \rightarrow R \rightarrow R} \text{ (SA-ARROW)}} \text{ (SA-ALL)}$$

Part II.

Polarized Higher Order Subtyping

4. Polarized system $F_{<}^\omega$:

In part I we have studied the system $F_{<}^\omega$ of higher-order subtyping with bounded quantification. The difficulty with that system is that, for type-checking with parametrized object types, it is too inflexible. In practice stronger subtyping is necessary. For example, we would like subtyping such as:

$$\text{List } A \leq \text{List } B \text{ if } A \leq B$$

under the assumption that lists are immutable, i.e. read-only. Consider if for example we have $\text{Int} \leq \text{Real}$, then we would reasonably expect $\text{List Int} \leq \text{List Real}$. As long as we only extract elements of the list, it is sound to "forget" that the elements of the list are integers, and "widen" their type to Real .

This is not allowed in the system $F_{<}^\omega$. (We have derived this rule as an example using the impredicative encoding of lists, but not using List as an abstract type constructor.)

On the other side, we must be careful not to allow subtyping such as

$$\text{Array } A \leq \text{Array } B \text{ if } A \leq B$$

because Array is read-write. The Java language requires a run-time type check on assignment to an array element, because it allows the subtyping rule.

This chapter extends the calculus $F_{<}^\omega$ by subtyping rules for a more general form of application, taking monotonicity information into account. Thus, besides the pointwise subtyping rule (S-APP) for applications, the system now allows to derive, for example, $\Gamma \vdash FG \leq FG' : \kappa'$ if $\Gamma \vdash G \leq G' : \kappa$ provided F is monotone. Besides monotone operators, the system formalizes also antitone and constant ones, and the ones without monotonicity information.

In the next chapter we introduce an algorithm for deciding equality and subtyping for this extended system. This time we follow another approach: the constructors are not fully normalized. Instead, we use weak head evaluation, i.e., the algorithm eliminates only the redexes necessary for going further. We claim that this algorithm is also sound and complete, but we could not prove its completeness with the method from chapters 2,3. For this reason, this part will be less formal than part I. We introduce the system and the algorithm, and explain its implementation in Haskell. The system and the algorithm extend Abel's in [Abe06b] for the whole system $F_{<}^\omega$ with bounded quantification.

4.1. Polarities

The following explanation is a citation from Abel's PhD Thesis, [Abe06c].

We aim to distinguish constructors with regard to their *monotonicity* or *variance*. For instance, the product constructor \times is *isotone* or *covariant* in both of its arguments. If one enlarges the type A or B , more terms inhabit $A \times B$. The

opposite behaviour is called *antitone* or *contravariant*. Two more scenarios are possible: the value $F A$ does not change when we modify A . Then F is called *constant* or *invariant*. Finally, a function F might not exhibit a uniform behaviour, it might grow or shrink with its arguments, or we just do not know how it behaves. This is the general case, we call it *mixed-variant*. Each of the behaviors is called a *polarity*, and abbreviated by one of the following four symbols:

$$\text{Pol} \ni p, q ::= \begin{array}{l} \circ \quad \text{mixed-variant} \\ | \quad + \quad \text{covariant} \\ | \quad - \quad \text{contravariant} \\ | \quad \top \quad \text{invariant} \end{array}$$

The polarities are related: Since "mixed-variant" just means that we do not have any information about the function, and we can always disregard our knowledge of variance, each function is mixed-variant. The inclusion order between the four sets of in-, co-, contra- and mixed-variant functions induces a partial information order \leq on Pol . It is given by:

$$\circ \leq p, p \leq \top, \quad \text{and} \quad p \leq q \quad \text{for all } p, q.$$

Polarity of composed functions. Let F, G be two functions such that the composition $F \circ G$ is well-defined. If F has polarity p and G has polarity q , we denote the polarity of the composed function $F \circ G$ by pq . Polarity composition is monotone: if one gets more information about F or G , certainly one cannot have less information about $F \circ G$. Then, if one of the functions is constant, so is their composition. Otherwise, if one of them is mixed-variant, the same holds for the composition. In the remaining cases, the composition is covariant if F and G have the same variance, otherwise it is contravariant. This yields the following multiplication table:

	\circ	$+$	$-$	\top
\circ	\circ	\circ	\circ	\top
$+$	\circ	$+$	$-$	\top
$-$	\circ	$-$	$+$	\top
\top	\top	\top	\top	\top

Inverse application of polarities. If $f(y) = py$ is the function which composes a polarity with p , what would be its inverse $g(x) = p^{-1}x$? It is possible to define g in such a way that f and g form a Galois connection, i.e.,

$$p^{-1}x \leq y \Leftrightarrow x \leq py.$$

This specification determines a unique solution for p^{-1} which is given by:

$$\begin{aligned} +^{-1}x &= x \\ -^{-1}x &= -x \\ \top^{-1}x &= \circ \\ \circ^{-1}\circ &= \circ \\ \circ^{-1}x' &= \top \quad (\text{for } x' \neq \circ). \end{aligned}$$

4.2. Kinds

The kinds of system $F_{<}^\omega$ are extended with polarities:

$$\begin{array}{l} \text{Kind} \ni \kappa ::= * \quad \text{types} \\ \quad \quad \quad | \quad p\kappa_1 \rightarrow \kappa_2 \quad p\text{-variant constructor transformers} \end{array}$$

A constructor of kind $p\kappa_1 \rightarrow \kappa_2$ is a p -variant function which maps constructors of kind κ_1 to constructors of kind κ_2 . Sometimes it is written as $\kappa_1 \xrightarrow{p} \kappa_2$.

Subkinding. The order on polarities induces an order $\kappa \leq \kappa'$ on kinds. We say that κ is a *subkind* of κ' or κ' is a *superkind* of κ . As usual, this shall mean that each constructor of kind κ is also of kind κ' . The subkinding relation is given inductively by the following rules:

$$\frac{}{* \leq *} \quad \frac{p' \leq p \quad \kappa'_1 \leq \kappa_1 \quad \kappa_2 \leq \kappa'_2}{p\kappa_1 \rightarrow \kappa_2 \leq p'\kappa'_1 \rightarrow \kappa'_2}$$

Subkinding is reflexive, transitive, and antisymmetric; hence, a proper partial order.

Constructors are defined as in Chapter 1, extended with some constants.

$$\text{Constr} \ni A, B, F, G, H, I, J ::= C \mid X \mid \lambda X F \mid F G \mid \forall X \leq G : \kappa. A$$

Signature. The constructor constants C are taken from a fixed *signature* Σ which contains at least the following constants together with their kinding:

$$\begin{array}{l} \rightarrow : * \xrightarrow{-} * \xrightarrow{+} * \quad \text{function space,} \\ \top : * \quad \text{largest type.} \end{array}$$

In a higher order system with type constructors distinguished by their variance, we do not need to treat the function space " \rightarrow " separately. It is just a type constructor, contravariant in its first argument, and covariant in the second. (s. the discussion in Chapter 1). We can now represent $A \rightarrow B$ as $C A B$ with $C = \rightarrow$. We continue to write \rightarrow infix.

Moreover, we redefine \top_κ by $\top_{\vec{\kappa} \rightarrow *} = \lambda \vec{Y} \top : \kappa_1 \xrightarrow{\top} \dots \xrightarrow{\top} \kappa_n \xrightarrow{\top} *$, where the lengths of \vec{Y} and $\vec{\kappa}$ coincide. We use the polarities for expressing that \top_κ is an invariant constructor.

4.3. Kinding

In this section, we present rules of kinding. The rules extend the kinding rules of $F_{<}^\omega$ by the treatment of polarities.

In the simply typed λ -calculus and in system F there are simple syntactic definitions of positive and negative occurrence of type variables in types. Let A be a type expression, viewed as a tree. A type variable X is said to occur *positively* in A if the path from the root to X takes the left branch of an \rightarrow -node an even number of times; otherwise, X occurs *negatively*. This simple syntactic criterion does not scale to a higher-order type system like $F_{<}^\omega$, hence, we will define "positively" and "negatively" via the kinding judgement.

Polarized contexts. A polarized context Γ fixes a bound G , a polarity p and a kind κ for each free variable X of a constructor F . If $p = +$, then X may only appear positively in F ; this ensures that λXF is an isotone function. Similarly, if $p = -$, then X may only occur negatively, and if $p = \circ$, then X may appear in both positive and negative positions.

$$\begin{array}{l} \text{PCxt } \ni \Gamma ::= \diamond \quad \text{empty context} \\ | \Gamma, X \leq G : p\kappa \quad \text{extended context with bounds } (X \notin \text{dom}(\Gamma)) \end{array}$$

The following restriction applies: $X \leq G : p\kappa \in \Gamma$ implies $p = \circ$ or $G = \top$. We write again $\Gamma, X : p\kappa$ as an abbreviation for $\Gamma, X \leq \top : p\kappa$. The default polarity is \circ .

Application of polarities to contexts. We define application $p\Gamma$ of a polarity p to a polarized context Γ . It composes p with every polarity assigned to a variable in Γ . The operation $\circ\Gamma$ removes all polarity information from a context, making it isomorphic to the kinding context in system F_{\leq}^{ω} from Chapter 1. Inverse application $p^{-1}\Gamma$ is defined analogously.

Well-formed contexts. Well-formed contexts $\Gamma \vdash$ are defined inductively as follows, mutually with the kinding judgement $\Gamma \vdash F : \kappa$.

$$\boxed{\Gamma \vdash}$$

$$\frac{}{\diamond \vdash} \text{ (C-EMPTY)} \quad \frac{\Gamma \vdash \quad \circ^{-1}\Gamma \vdash G : \kappa}{\Gamma, X \leq G : p\kappa \vdash} \text{ (C-BOUND)}$$

Kinding.

$$\boxed{\Gamma \vdash F : \kappa}$$

$$\frac{\Gamma \vdash \quad C : \kappa \in \Sigma}{\Gamma \vdash C : \kappa} \text{ (KIND-C)} \quad \frac{\Gamma \vdash \quad X \leq G : p\kappa \in \Gamma \quad p \leq +}{\Gamma \vdash X : \kappa} \text{ (KIND-VAR-BOUND)}$$

$$\frac{\Gamma, X : p\kappa \vdash F : \kappa'}{\Gamma \vdash \lambda XF : p\kappa \rightarrow \kappa'} \text{ (KIND-}\lambda\text{)} \quad \frac{\Gamma \vdash F : p\kappa \rightarrow \kappa' \quad p^{-1}\Gamma \vdash G : \kappa}{\Gamma \vdash FG : \kappa'} \text{ (KIND-APP)}$$

$$\frac{\Gamma, X \leq G : \circ\kappa \vdash A : *}{\Gamma \vdash \forall X \leq G : \kappa. A : *} \text{ (KIND-}\forall\text{)} \quad \frac{\Gamma \vdash F : \kappa \quad \kappa \leq \kappa'}{\Gamma \vdash F : \kappa'} \text{ (KIND-SUB)}$$

To understand the kinding rules, it is useful to think of F in the context $\Gamma = \vec{X} \leq \vec{G} : p\vec{\kappa}$ as a function $F(\vec{X})$. The judgement $\Gamma \vdash F : \kappa$ would be valid if both F has kind κ and F grows whenever the positive arguments to F grow, the negative shrink and mixed-variant stay fixed. Moreover, F stays fixed when the constant arguments grow or shrink. Hence, rule (KIND-VAR) can only allow non-negative variables to be fetched from the context.

For understanding the rule (KIND-APP) we consider the cases $p = +, -, \circ, \top$ separately.

$$\frac{\Gamma \vdash F : +\kappa \rightarrow \kappa' \quad \Gamma \vdash G : \kappa}{\Gamma \vdash FG : \kappa'}$$

In the case $p = +$, F is a monotone function, hence, whenever its argument G grows, the application FG grows as well. Hence, the polarity of the variables in G match the polarity of the variables in FG .

$$\frac{\Gamma \vdash F: -\kappa \rightarrow \kappa' \quad -\Gamma \vdash G: \kappa}{\Gamma \vdash FG: \kappa'}$$

If F is antitone, the application FG will grow if G shrinks. Hence, the polarity of the variables in G must be opposite ($-\Gamma$) to the one of the variables in the application (Γ). Imagine a variable X appearing negatively in G . If it grows, G will shrink, hence FG will grow. Therefore X appears positively in the application.

$$\frac{\Gamma \vdash F: \circ\kappa \rightarrow \kappa' \quad \circ^{-1}\Gamma \vdash G: \kappa}{\Gamma \vdash FG: \kappa'}$$

In the mixed-variant case, we do not know how F behaves if we modify its arguments. To satisfy our informal semantics, FG needs to grow if we grow the variables declared in Γ to be positive, shrink the negative variables, and leave the mixed-variant ones fixed. If one of the positive or negative variables appeared in G , the argument to F would change which would result in an unpredictable shift of the value of FG . If we want to verify that FG grows, we need to ensure that no positive or negative variables occur in G . This is done by changing the polarity of all positive or negative variables from the context to \top through the operation $\circ^{-1}\Gamma$.

$$\frac{\Gamma \vdash F: \top\kappa \rightarrow \kappa' \quad \top^{-1}\Gamma \vdash G: \kappa}{\Gamma \vdash FG: \kappa'}$$

If F is constant, F remains fixed when we modify its arguments. Hence, a change in G does not affect FG . Therefore, variables may appear in G arbitrarily. Thus, when kinding G , we set the polarity of all variables to \circ by the operation $\top^{-1}\Gamma$.

Example 4.3.1 (Derived rule for function space) *The following rule is derivable:*

$$\frac{-\Gamma \vdash A: * \quad \Gamma \vdash B: *}{\Gamma \vdash A \rightarrow B: *}$$

In the rule (KIND- \forall) the bound variable is added to the context with a mixed-variant polarity. Moreover, in (C-BOUND):

$$\frac{\Gamma \vdash \quad \circ^{-1}\Gamma \vdash G: \kappa}{\Gamma, X \leq G: p\kappa \vdash} \text{ (C-BOUND)}$$

we "erase" all positive or negative variables of G by $\circ^{-1}\Gamma$. This is because we are in the Kernel-variant of F_{\leq}^{ω} . Consider the following example:

$$\frac{X: +* \vdash \quad X: +* \vdash X: *}{X: +, Y \leq X: \circ* \vdash}$$

The variable X is positive in X and, with a naive (C-BOUND) rule, X would appear positive in the bounded quantification $\forall Y \leq X: *.Y$ as well.

$$\frac{X: +*, Y \leq X: \circ* \vdash Y: *}{X: +* \vdash \forall Y \leq X: *.Y: *}$$

That would mean that $\forall Y \leq X: *.Y$ would be monotone in X , but it is antitone. Moreover, we treat bounded quantification in the kernel variant:

$$\frac{\Gamma, X \leq G : \kappa \vdash A \leq A' : *}{\Gamma \vdash \forall X \leq G. A \leq \forall X < G. A' : *} \quad (\text{LEQ-}\forall)$$

Because the bounds need to be equal, they should not contain positive or negative variables. Otherwise the following would be possible:

$$Y : -* \vdash \forall X \leq Y. X \leq \forall X < Y. X$$

thus, if we substitute Y with $G' \leq G$, which would be allowed because of the negative polarity of Y , we would obtain:

$$\forall X \leq G. X \leq \forall X \leq G'. X$$

This is semantically correct, but it is not allowed in the kernel-variant.

4.4. Equality

In this section we extend the $\beta\eta$ -equality of the system F_{\leq}^{ω} for the treatment of polarities. The most important extension is the addition of a new axiom (EQ- \top), which formalizes the fact that constant functions do not depend on its arguments.

$$\boxed{\Gamma \vdash F = F' : \kappa}$$

Axioms.

$$\frac{\Gamma, X : p\kappa \vdash F : \kappa' \quad p^{-1}\Gamma \vdash G : \kappa}{\Gamma \vdash (\lambda X F) G = [G/X]F : \kappa'} \quad (\text{EQ-}\beta)$$

$$\frac{\Gamma \vdash F : p\kappa \rightarrow \kappa'}{\Gamma \vdash (\lambda X. FX) = F : p\kappa \rightarrow \kappa'} \quad X \notin \text{FV}(F) \quad (\text{EQ-}\eta)$$

$$\frac{\Gamma \vdash F : \top\kappa \rightarrow \kappa' \quad \top^{-1}\Gamma \vdash G : \kappa \quad \top^{-1}\Gamma \vdash G' : \kappa}{\Gamma \vdash FG = FG' : \kappa'} \quad (\text{EQ-}\top)$$

Congruence rules and subsumption.

$$\frac{\Gamma \vdash C : \kappa \in \Sigma}{\Gamma \vdash C = C : \kappa} \quad (\text{EQ-C}) \quad \frac{\Gamma \vdash X \leq G : p\kappa \in \Gamma \quad p \leq +}{\Gamma \vdash X = X : \kappa} \quad (\text{EQ-VAR-BOUND})$$

$$\frac{\Gamma, X : p\kappa \vdash F = F' : \kappa'}{\Gamma \vdash \lambda X F = \lambda X F' : p\kappa \rightarrow \kappa'} \quad (\text{EQ-}\lambda) \quad \frac{\circ^{-1}\Gamma \vdash G = G' : \kappa \quad \Gamma, X \leq G : \circ\kappa \vdash A = A' : *}{\Gamma \vdash \forall X \leq G : \kappa. A = \forall X \leq G' : \kappa. A' : *} \quad (\text{EQ-}\forall)$$

$$\frac{\Gamma \vdash F = F' : p\kappa \rightarrow \kappa' \quad p^{-1}\Gamma \vdash G = G' : \kappa}{\Gamma \vdash FG = F'G' : \kappa'} \quad (\text{EQ-APP})$$

$$\frac{\Gamma \vdash F = F' : \kappa \quad \kappa \leq \kappa'}{\Gamma \vdash F = F' : \kappa'} \quad (\text{EQ-SUB})$$

Symmetry and transitivity.

$$\frac{\Gamma \vdash F = F' : \kappa}{\Gamma \vdash F' = F : \kappa} \text{ (EQ-SYM)} \quad \frac{\Gamma \vdash F_1 = F_2 : \kappa \quad \Gamma \vdash F_2 = F_3 : \kappa}{\Gamma \vdash F_1 = F_3 : \kappa} \text{ (EQ-TRANS)}$$

Admissible rules for equality

$$\frac{\Gamma \vdash F : \kappa}{\Gamma \vdash F = F : \kappa} \text{ (EQ-REFL)}$$

In the rule (EQ-APP) we have to modify the polarities of the variables in G and G' in the same way as in the application rule of the kinding judgement: We inverse-apply p to the context Γ .

4.5. Subtyping

In this section, we specify subtyping for constructors of polarized kinds.

$$\boxed{\Gamma \vdash F \leq F' : \kappa}$$

Reflexivity, transitivity.

$$\frac{\Gamma \vdash F = F' : \kappa}{\Gamma \vdash F \leq F' : \kappa} \text{ (LEQ-REFL)} \quad \frac{\Gamma \vdash F_1 \leq F_2 : \kappa \quad \Gamma \vdash F_2 \leq F_3 : \kappa}{\Gamma \vdash F_1 \leq F_3 : \kappa} \text{ (LEQ-TRANS)}$$

Abstraction.

$$\frac{\Gamma, X : p\kappa \vdash F \leq F' : \kappa'}{\Gamma \vdash \lambda X F \leq \lambda X F' : p\kappa \rightarrow \kappa'} \text{ (LEQ-}\lambda\text{)}$$

Application. There are two kinds of congruence rules for application: one kind states that if functions F and F' are in the subtyping relation, so are their values $F G$ and $F' G$ at a certain argument G .

$$\frac{\Gamma \vdash F \leq F' : p\kappa \rightarrow \kappa' \quad p^{-1}\Gamma \vdash G : \kappa}{\Gamma \vdash F G \leq F' G : \kappa'} \text{ (LEQ-FUN)}$$

The other kind of rules concern the opposite case: If F is a function and two arguments G and G' are in a subtyping relation, so are the values $F G$ and $F G'$ of the function at these arguments. However such a relation can only exist if F is either covariant or contravariant.

$$\frac{\Gamma \vdash F : +\kappa \rightarrow \kappa' \quad \Gamma \vdash G \leq G' : \kappa}{\Gamma \vdash F G \leq F G' : \kappa'} \text{ (LEQ-ARG+)}$$

$$\frac{\Gamma \vdash F : -\kappa \rightarrow \kappa' \quad -\Gamma \vdash G' \leq G : \kappa}{\Gamma \vdash F G \leq F G' : \kappa'} \text{ (LEQ-ARG-)}$$

Quantification, Bounds and Top.

$$\frac{\Gamma, X \leq G : \circ\kappa \vdash A \leq A' : *}{\Gamma \vdash \forall X \leq G : \kappa. A \leq \forall X \leq G : \kappa. A' : *} \quad (\text{LEQ-}\forall)$$

$$\frac{\Gamma \vdash \quad X \leq F : p\kappa \in \Gamma \quad p \leq +}{\Gamma \vdash X \leq F : \kappa} \quad (\text{LEQ-BOUND}) \quad \frac{\Gamma \vdash A : *}{\Gamma \vdash A \leq \top : *} \quad (\text{LEQ-}\top)$$

Subsumption.

$$\frac{\Gamma \vdash F \leq F' : \kappa \quad \kappa \leq \kappa'}{\Gamma \vdash F \leq F' : \kappa'} \quad (\text{LEQ-SUB})$$

Admissible rules for subtyping.

$$\frac{\Gamma \vdash F \leq F' : \circ\kappa \rightarrow \kappa' \quad \circ^{-1}\Gamma \vdash G = G' : \kappa}{\Gamma \vdash FG \leq F'G' : \kappa'} \quad (\text{LEQ-APP}\circ)$$

$$\frac{\Gamma \vdash F \leq F' : +\kappa \rightarrow \kappa' \quad \Gamma \vdash G \leq G' : \kappa}{\Gamma \vdash FG \leq F'G' : \kappa'} \quad (\text{LEQ-APP+})$$

$$\frac{\Gamma \vdash F \leq F' : -\kappa \rightarrow \kappa' \quad -\Gamma \vdash G' \leq G : \kappa}{\Gamma \vdash FG \leq F'G' : \kappa'} \quad (\text{LEQ-APP-})$$

$$\frac{\Gamma \vdash F \leq F' : \top\kappa \rightarrow \kappa' \quad \top^{-1}\Gamma \vdash G : \kappa \quad \top^{-1}\Gamma \vdash G' : \kappa}{\Gamma \vdash FG \leq F'G' : \kappa'} \quad (\text{LEQ-APP}\top)$$

5. Algorithmic Polarized Subtyping

In this chapter, we present an algorithm for deciding whether two well-kinded constructors are equal or related by subtyping. The idea is to first weak-head normalize the constructors under consideration and then compare their head symbols. If they are related, one continues to recursively compare the subcomponents, otherwise subtyping fails.

5.1. Weak head evaluation

Weak head normal forms $W \in \text{Val}$ are given by the grammar:

$$\begin{array}{ll} \text{Ne} \ni N & ::= C \mid X \mid NG & \text{neutral constructors} \\ \text{Val} \ni V, W & ::= N \mid \lambda XF \quad \mid \forall X \leq G : \kappa. A & \text{weak head values} \end{array}$$

Weak head evaluation $F \searrow W$, a big-step call-by-name operational semantics, is defined inductively by the following rules:

$$\begin{array}{c} \frac{}{C \searrow C} \text{ (EVAL-C)} \quad \frac{}{X \searrow X} \text{ (EVAL-VAR)} \quad \frac{}{\lambda XF \searrow \lambda XF} \text{ (EVAL-LAM)} \\ \\ \frac{}{\forall X \leq G : \kappa. A \searrow \forall X \leq G : \kappa. A} \text{ (EVAL-}\forall\text{)} \\ \\ \frac{F \searrow N}{FG \searrow NG} \text{ (EVAL-APP-NE)} \quad \frac{F \searrow \lambda XF' \quad [G/X]F' \searrow W}{FG \searrow W} \text{ (EVAL-APP-}\beta\text{)} \end{array}$$

5.2. Algorithm

We are ready to define the subtyping algorithm. Note that at any point during subtyping checking we may require kinding information. For example, consider checking $XG \leq XG'$. If X is covariant, we need to continue with $G \leq G'$, but if X is contravariant, the next step would be checking $G' \leq G$. Hence, the algorithm needs both context Γ and kind κ of the two considered constructors as additional input.

The general form of the algorithmic subtyping judgement is defined by $\Gamma \vdash_a F \leq^q F' \Leftarrow \kappa$, a judgement defined inductively below. The polarity q codes the relation that we seek to establish between F and F' : If $q = \circ$, we expect them to be equal, if $q = +$, we expect $F \leq F'$, and if $q = -$, then the other way round. Finally if $q = \top$, then F and F' need not be related, and the algorithm succeeds immediately.

The judgements for algorithmic subtyping $\Gamma \vdash_a N \leq^q N' \Rightarrow \kappa$ for neutral constructors, $\Gamma \vdash_a W \leq^q W' : \kappa$ for weak head values and $\Gamma \vdash_a F \leq^q F' \Leftarrow \kappa$ kind-directed are defined inductively by the following rules.

Algorithmic subtyping and equality.

Inference mode, $q \in \{+, -, \circ\}$ $\boxed{\Gamma \vdash_a N \leq^q N' \Rightarrow \kappa}$

$$\frac{(C:\kappa) \in \Sigma}{\Gamma \vdash_a C \leq^q C \Rightarrow \kappa} \text{ (AL-C)} \quad \frac{(X \leq G:p\kappa) \in \Gamma \quad p \leq +}{\Gamma \vdash_a X \leq^q X \Rightarrow \kappa} \text{ (AL-VAR-BOUND)}$$

$$\frac{\Gamma \vdash_a N \leq^q N' \Rightarrow p\kappa \rightarrow \kappa' \quad p^{-1}\Gamma \vdash_a G \leq^{pq} G' \Leftarrow \kappa}{\Gamma \vdash_a NG \leq^q N'G' \Rightarrow \kappa'} \text{ (AL-APP-NE)}$$

Weak head mode, $q \in \{+, -, \circ\}$ $\boxed{\Gamma \vdash_a W \leq^q W' : \kappa}$

$$\frac{\Gamma \vdash_a N \leq^q N' \Rightarrow \kappa}{\Gamma \vdash_a N \leq^q N' : * } \text{ (AL-NE)}$$

$$\overline{\Gamma \vdash_a \top \leq^- W : * } \text{ (AL-TOP-)} \quad \overline{\Gamma \vdash_a W \leq^+ \top : * } \text{ (AL-TOP+)} \quad \overline{\Gamma \vdash_a \top \leq^\circ \top : * } \text{ (AL-TOP}_\circ\text{)}$$

$$\frac{\circ^{-1}\Gamma \vdash_a G \leq^\circ G' \Leftarrow \kappa \quad \Gamma, X \leq G:\circ\kappa \vdash_a A \leq^q A' \Leftarrow *}{\Gamma \vdash_a \forall X \leq G:\kappa. A \leq^q \forall X \leq G':\kappa. A' : * } \text{ (AL-}\forall\text{)}$$

$$\frac{(X \leq G:\circ\kappa) \in \Gamma \quad \Gamma \vdash_a V \leq^+ W : * \quad G\vec{F} \searrow V}{\Gamma \vdash_a X\vec{F} \leq^+ W : * } W \neq X\vec{F}' \text{ (AL-BOUND}_+\text{)}$$

$$\frac{(X \leq G:\circ\kappa) \in \Gamma \quad \Gamma \vdash_a W \leq^- V : * \quad G\vec{F} \searrow V}{\Gamma \vdash_a W \leq^- X\vec{F} : * } W \neq X\vec{F}' \text{ (AL-BOUND}_-\text{)}$$

Check mode, $q \in \{+, -, \circ\}$ (*kind directed*) $\boxed{\Gamma \vdash_a F \leq^q F' \Leftarrow \kappa}$

$$\frac{\Gamma, X:p\kappa \vdash_a FX \leq^q F'X \Leftarrow \kappa'}{\Gamma \vdash_a F \leq^q F' \Leftarrow p\kappa \rightarrow \kappa'} \text{ (AL-FUN)} \quad X \text{ new}$$

$$\frac{F \searrow W \quad F' \searrow W' \quad \Gamma \vdash_a W \leq^q W' : *}{\Gamma \vdash_a F \leq^q F' \Leftarrow * } \text{ (AL-WEAK)}$$

Check mode, $q \in \{\top\}$

$$\overline{\Gamma \vdash_a F \leq^\top F' \Leftarrow \kappa} \text{ (AL-ANY)}$$

They are deterministic and can be directly implemented as an algorithm (apply the rules backwards). All three judgements take the context Γ , the polarity q , and the two constructors as input. Judgement $\Gamma \vdash_a N \leq^q N' \Rightarrow \kappa$ produces kind κ if it succeeds, whereas the judgement $\Gamma \vdash_a F \leq^q F' \Leftarrow \kappa$ takes κ as an additional input and either succeeds or fails. The direction of the double arrow indicates the flow of the kinding information out of (\Leftarrow) or into (\Rightarrow) κ .

Special cases of (AL-APP-NE)

$$\frac{\Gamma \vdash_{\mathbf{a}} N \leq^+ N' \Rightarrow p\kappa \rightarrow \kappa' \quad p^{-1}\Gamma \vdash_{\mathbf{a}} G \leq^p G' \Leftarrow \kappa}{\Gamma \vdash_{\mathbf{a}} NG \leq^+ N' G' \Rightarrow \kappa'} \quad (\text{AL-APP-NE})_{q=+}$$

$$\frac{\Gamma \vdash_{\mathbf{a}} N \leq^- N' \Rightarrow p\kappa \rightarrow \kappa' \quad p^{-1}\Gamma \vdash_{\mathbf{a}} G \leq^{-p} G' \Leftarrow \kappa}{\Gamma \vdash_{\mathbf{a}} NG \leq^- N' G' \Rightarrow \kappa'} \quad (\text{AL-APP-NE})_{q=-}$$

$$\frac{\Gamma \vdash_{\mathbf{a}} N \leq^{\circ} N' \Rightarrow \top\kappa \rightarrow \kappa' \quad \circ\Gamma \vdash_{\mathbf{a}} G \leq^{\top} G' \Leftarrow \kappa}{\Gamma \vdash_{\mathbf{a}} NG \leq^{\circ} N' G' \Rightarrow \kappa'} \quad (\text{AL-APP-NE})_{q=\circ, p=\top}$$

$$\frac{\Gamma \vdash_{\mathbf{a}} N \leq^{\circ} N' \Rightarrow p\kappa \rightarrow \kappa' \quad p^{-1}\Gamma \vdash_{\mathbf{a}} G \leq^{\circ} G' \Leftarrow \kappa}{\Gamma \vdash_{\mathbf{a}} NG \leq^{\circ} N' G' \Rightarrow \kappa'} \quad (\text{AL-APP-NE})_{q=\circ, p \neq \top}$$

5.3. Implementation in Haskell

The algorithm has been implemented in Haskell. Polarities and kinds have been implemented following the grammar and definitions from above. When implementing lambda terms the implementation and manipulation of variable binding is a key issue. There are many approaches for handling with it. We use for this implementation the "locally nameless" representation. In locally nameless representation, bound variables are represented by de Bruijn indices while free variables are represented by names. This mixed representation combines the benefits of both approaches, avoiding difficulties associated with alpha-conversion by ensuring that each alpha-equivalence class of terms has a unique representation, while supporting formal reasoning that closely follows informal practice.

The algorithm itself is implemented by three mutually recursive functions. Each of them implements one of the modes: inference mode, weak head mode and check mode. The algorithm returns a derivation tree, whose string representation is very close to the paper representation. This makes it very suitable for testing. A number of examples has been tested with it and all of them were successful. More details about the implementation can be found in the appendix.

Testing

The algorithm has been tested successfully with the following judgements. The result is in every case the expected one. Let $\kappa_1 = * \xrightarrow{+} *$.

Judgement	Result
1. $\vdash_a \lambda X X \leq^+ \lambda X. \top \Leftarrow * \xrightarrow{+} *$	True
2. $\vdash_a \lambda X. \forall Y \leq (\lambda X \lambda Y. X). Y X \leq^+ \lambda X. \forall Y \leq (\lambda X. X). Y \Leftarrow * \xrightarrow{+} *$	False
3. $\vdash_a \lambda X \lambda Y. X \rightarrow Y \leq^\circ \lambda X \lambda Y. X \rightarrow Y \Leftarrow * \xrightarrow{-} * \xrightarrow{+} *$	True
4. $\vdash_a \forall F \leq (\lambda H \lambda Y H). \forall G \leq (\lambda X X). \forall Z \leq \top_{\kappa_1}. \forall A \leq \top. F G Z A \leq^+$ $\forall F \leq (\lambda H \lambda Y H). \forall G \leq (\lambda X X). \forall Z \leq \top_{\kappa_1}. \forall A \leq \top. F G Z A \Leftarrow *$ the head variables are equal in both terms, thus, (AL-BOUND ₊) is not called.	True
5. $\vdash_a \forall 2 \leq (\lambda F \lambda X. F(FX)). \forall \text{ld} \leq (\lambda X X). 2(2(2(2(2 \text{ld})))) \leq^+$ $\forall 2 \leq (\lambda F \lambda X. F(FX)). \forall \text{ld} \leq (\lambda X X). \text{ld} \Leftarrow *$ in this case the head variables are not equal, so (AL-BOUND ₊) is called many times and the derivation is very long, but it succeeds. The reason is $(\lambda F \lambda X. F(FX))(\lambda X X) =_\beta (\lambda X. (\lambda X X)((\lambda X X)X)) =_\beta (\lambda X. X)(\lambda X X) =_\beta \lambda X X$ so using the bounds $2 \text{ld} =_\beta \text{ld}$	True

Conclusions

In this thesis we have presented algorithmic subtyping rules for the system $F_{<}^{\omega}$, a calculus for higher order polymorphism with subtyping and bounded quantification. The metatheory of this and similar systems has been studied in the literature, among others by (Compagnoni and Goguen [CG03]; Pierce and Steffen [PS97]), but their proofs of completeness of algorithmic subtyping are complicated and long. They used model construction or strong normalization theorems. The goal of this thesis was to prove completeness in an easier way, using syntactical proofs.

For this purpose we have reviewed the system $F_{<}^{\omega}$ with kinding, equality and subtyping, and formally proved properties of this system like validity of kinding, equality and subtyping, wakening, inversion, etc. Afterwards we defined a normalization function with the help of a new form of substitution: hereditary substitution, which preserves normal forms. The so defined normalizer computes η -long β -normal forms, and we proved its soundness and completeness.

Afterwards, we have recapitulated the usual algorithm for deciding subtyping for $F_{<}^{\omega}$, which operates on normal type constructors. Soundness was easy to prove, by induction on derivations. For proving completeness we proved lemmas stating the reflexivity and transitivity of the algorithmic rules, and a lemma about substitution, the key step for showing completeness for the application rule (S-APP). This is the step where the most simplifications in the proofs have been achieved.

We also have provided a proof of the termination of the algorithm, using the idea from Pierce and Steffen [PS97] of Γ -reduction. The idea of the proof is to characterize inductively constructors in η -long β -normal form, prove the soundness and completeness of this characterization and finally prove the termination of the algorithm when applied to those inductively defined normal constructors.

In part II we have extended the system $F_{<}^{\omega}$ with polarities in the kinds. We have given a detailed explanation of the informal semantics of the system, which justify the kinding and subtyping rules. Moreover, we give an algorithm for deciding equality and subtyping for this system. This kind directed algorithm is an extension for bounded quantification of Abel's algorithm in [Abe06b]. Our attempt to prove the completeness of this algorithm failed. Nevertheless we believe that the direct extension of our first algorithm to the polarized system would succeed.

Further work. The contribution of this thesis in the research context is the simplification of the completeness proofs of algorithmic subtyping for the system $F_{<}^{\omega}$. These syntactical proofs are much easier to understand than the former ones and they are also suitable for formalization in theorem provers like Twelf. Therefore, a good complementation of the research in this thesis would be to perform such formalizations. Moreover, there are extensions of the system $F_{<}^{\omega}$, like the polarized version we investigated as well, and the system $\mathcal{F}_{<}^{\omega}$ with bounded operators $\lambda X \leq G.F$ investigated by Compagnoni and Goguen [CG03]. Thus, an extension of this thesis could be the simplification of the completeness proofs for these other systems. For the

polarized system, we found difficulties with the proof, and we conjecture the problems are related to the weak head normalization. If we fully normalize the constructors instead, like we did for the system without polarities, we conjecture that the proofs will succeed. This is indeed worth to be investigated in the future.

Appendix

Implementation in Haskell

Polarities

```
module Pol where
```

We define the four polarities: $+$, $-$, \circ , \top .

```
data Pol = Co | Contra | Inv | Mix
  deriving (Eq)
```

Afterwards we define the order of the polarities, i.e. $\circ \leq p, p \leq p, p \leq \top$.

```
instance Ord Pol where
  Mix <= _ = True
  _ <= Inv = True
  p <= q | p == q = True
  _ <= _ = False
```

Next, we define polarity composition. We just implement the defined composition matrix.

```
comp :: Pol -> Pol -> Pol
```

We write composition in infix form, i.e. $p \# q$.

```
infixr 5 #
(#) = comp
```

With the composition we can define the inversion.

```
inv :: Pol -> Pol -> Pol
```

The implementation also follows directly from the definition.

Kinds

```
module Kind where
  import Pol
```

Kinds are either $*$ or $p\kappa_1 \rightarrow \kappa_2$.

```
data Kind =
  TYPE | ARR PKind Kind
  deriving Eq
```

PKinds are polarized kinds, actually a pair of a polarity and a kind.

```
data PKind =
  PK Pol Kind
  deriving (Eq)
```

Constructors

```
module Constr where
  import Kind
  import Pol
```

Ind and *Name* are just other names for *Int* and *String*.

```
type Ind = Int
type Name = String
```

LeqConstr is a data type for implementing a bound (a constructor) and its kind.

```
data LeqConstr =
  Leq Constr Kind
  deriving Eq
```

The constructors are implemented by the data type *Constr*. Constants are implemented by a unique name for looking up in the signature Σ . There are two data constructors for variables, *Ref Ind* and *Var Name*. The reason for it is that we use the approach "locally nameless" for implementing variable binding. A bound variable is represented by *Ref n*, where *n* is a natural number pointing to the position of its binder from right to left starting by 0. For example: $\lambda X.X$ is represented by *Lam (Ref 0)* and $\lambda X \lambda Y.X$ is represented by *Lam (Lam (Ref 1))*. Free variables are represented by a unique name for looking up in the context Γ . The bounded quantification is implemented as a pair of a *LeqConstr* and a *Constr*.

```
data Constr = Const Name | Ref Ind | Var Name | Lam Constr |
  App Constr Constr | Forall LeqConstr Constr | Top
  deriving Eq
```

whd is the implementation of the weak head evaluation from above.

```
whd :: Constr → Constr
```

Contexts and Derivation Trees

```
module Cxt where
  import Kind
```

```

import Pol
import Constr

```

The data type *PCxtItem* represents an item in the context. For simplicity we represent separately variables with trivial bounds (\top_κ) as a name with a polarized kind and variables with real bounds as a name and a bound with its polarized kind.

```

data PCxtItem =
  Cxt Name PKind |
  Cxtb Name LeqConstr
deriving Eq

```

The context is then a list of context items.

```

type PCxt = [PCxtItem]

```

Similarly, we define an item in the signature as a name with a kind.

```

data CCxtItem =
  C Name Kind
deriving Eq

```

And the signature is a list of those items.

```

type CCxt = [CCxtItem]

```

This is our signature with the function space " \rightarrow " of kind $* \rightarrow * \xrightarrow{\pm} *$.

```

sigma :: CCxt
sigma = [(C "->" ((PK Contra k0) ==> k1))]

```

Next, we define auxiliary data types and functions for representing the trees of derivations returned by the algorithm.

The data type *Mod* represents one of the three modes of the algorithm: check mode, weak head mode or inference mode, and one of the rules of the respective mode.

```

data Mod = Check Rule | Whd Rule | Inf Rule
deriving Eq

```

And these are the rules, for combining with the modes.

```

data Rule = INV_R | WEAK | FUN | BOUND |
  FORALL | TOP | NE | APP_NE | VAR | CONST
deriving Eq

```

The next definition is the data structure for representing derivation trees. There are binary nodes for representing the derivation steps given by the rules (AL-APP-NE) and (AL- \forall), unary nodes for the rules (AL-WEAK), (AL-FUN), (AL-BOUND₊), (AL-BOUND₋) and (AL-NE), and leaves for the rules (AL-TOP₊), (AL-TOP₋), (AL-TOP_o), (AL-C) and (AL-VAR-BOUND). Moreover, the data constructor *Nil* represents a leaf that does not correspond to any rule, i.e. a wrong derivation.

For deciding $\Gamma \vdash_a F \leq^q F' \Leftarrow \kappa$ the algorithm returns a derivation tree. The claim holds if all the leaves of the tree are an instance of one of the rules, i.e. a *Leaf*. If there is a *Nil* leaf, then the claim does not hold. It is easy to build a function that takes the derivation tree and returns *True* or *False* by visiting all the leaves of the tree, for example with a depth first search. It is also possible to implement the algorithm in such a way that it returns directly *True* or *False* instead of a tree. We prefer to return a tree, though, because this way we get more information from the algorithm: an answer and a justification for it at the same time.

The information saved in the nodes is the context Γ , the polarity q , two constructors, their kind, the mode and rule information and the subtrees.

```

data DerTree =
  BiNode PCxt Pol Constr Constr Kind Mod DerTree DerTree |
  UnNode PCxt Pol Constr Constr Kind Mod DerTree |
  Leaf | Nil
  deriving (Eq)

```

Algorithm

The algorithm consists of three mutual recursive functions:

- *inf mod* implements the inference mode
- *whd mod* implements the weak head mode
- *check mod* implements the check mode

```

module Main where
  import Constr
  import Pol
  import Cxt
  import Kind
  import Data.Maybe

```

Inference mode ($\Gamma \vdash_a N \leq^q N' \Rightarrow \kappa$). The function *inf mod* takes a context, a polarity and two constructors as arguments, and returns a derivation tree. In the specification of the algorithm, this part of the algorithm returns a kind. In this implementation the kind is just part of the tree, and can be extracted from it.

$$inf_mod :: PCxt \rightarrow Pol \rightarrow Constr \rightarrow Constr \rightarrow DerTree$$

Inference mode, handling of variables (AL-VAR). If the constructors are equal variables, and their common name can be found in the context with *lookup var* with a polarity less than +, then the result is a unary node recording the context, the polarity, the constructors, the kind from the variable in the context, and *Leaf* as subtree. Otherwise the subtree is *Nil*.

$$\begin{aligned}
&inf_mod\ gamma\ q\ (Var\ a)\ (Var\ a') \\
&\quad | (a \equiv a') \wedge (jpk \neq Nothing) \wedge (p \leq Co) = tree\ Leaf\ k
\end{aligned}$$

| *otherwise* = *tree Nil k0*
where
jpg = *lookup_var a gamma*
pk = (*fromJust jpg*)
p = (*pol pk*)
k = (*kind pk*)
tree = $\lambda t k \rightarrow \text{UnNode } \textit{gamma } q \textit{ (Var } a \textit{) (Var } a' \textit{) } k \textit{ (Inf VAR) } t$

Inference mode, handling of constants (AL-C). Very similar to the variable case.

inf_mod gamma q (Const a) (Const a')
| (*a* \equiv *a'*) \wedge (*jk* \neq *Nothing*) = *tree Leaf k*
| *otherwise* = *tree Nil k0*
where
jk = *lookup_const a sigma*
k = (*fromJust jk*)
tree = $\lambda t k \rightarrow \text{UnNode } \textit{gamma } q \textit{ (Const } a \textit{) (Const } a' \textit{) } k \textit{ (Inf CONST) } t$

Inference mode, handling of application (AL-APP-NE). In this case the constructors are applications, where the first constructors are neutral. Then the algorithm calls recursively the inference mode with the neutral constructors for inferring a kind $p\kappa_1 \rightarrow \kappa_2$, that is extracted from the result tree. Afterwards the algorithm calls the check mode with the second arguments of the application, and the inversed context $p^{-1}\Gamma$, the polarity *pq* and the kind κ_1 . Then it returns a binary node recording the two application constructors, the context, the polarity, the kind κ_2 and the subtrees returned by the recursive calls.

inf_mod gamma q (App n g) (App n' g')
| (*isNeutral n*) \wedge (*isNeutral n'*) =
BiNode gamma q (App n g) (App n' g') k2 (Inf APP_NE) node1 node2
where
node1 = (*inf_mod gamma q n n'*)
k = *fromJust (kind_tree node1)*
k2 = *arr2 k*
k1 = *kind (arr1 k)*
p = *pol (arr1 k)*
igamma = (*inv_cxt p gamma*)
p' = *p # q*
node2 = (*check igamma p' g g' k1*)

Weak head mode ($\Gamma \vdash_a W \leq^q W' : *$). In the weak head mode the kind information is not important, because it is always $*$.

whd_mod :: *PCxt* \rightarrow *Pol* \rightarrow *Constr* \rightarrow *Constr* \rightarrow *DerTree*

Weak head mode, handling of \top : $W \leq^+ \top$, $\top \leq^- W$, $\top \leq^\circ \top$ (AL-TOP_q). The implementation is straightforward.

whd_mod gamma Contra Top c2 = *UnNode gamma Contra Top c2 TYPE (Whd TOP) Leaf*
whd_mod gamma Co c1 Top = *UnNode gamma Co c1 Top TYPE (Whd TOP) Leaf*
whd_mod gamma Mix Top Top = *UnNode gamma Mix Top Top TYPE (Whd TOP) Leaf*

Weak head mode, handling of bounded quantification (AL- \forall). The two constructors are $\forall X \leq G_1 : \kappa_1. A_1$ and $\forall X \leq G_2 : \kappa_2. A_2$. First, the kind of the bounds must coincide, otherwise returns *Nil* as the subtree. Then, the algorithm calls recursively the check mode with A and A' and the same polarity q and kind $*$, but with an extended context by the bounded item $X \leq G : \kappa_1$, and it unbinds the variable X in A and A' . Moreover, it also checks recursively that the bounds G and G' are equal (polarity \circ) with kind κ_1 .

```

whd_mod gamma q (Forall (Leq g k1) a) (Forall (Leq g' k2) a')
| (k1  $\neq$  k2) = UnNode gamma q
                (Forall (Leq g k1) a)
                (Forall (Leq g' k2) a')
                TYPE (Whd FORALL) Nil
| otherwise = BiNode gamma q
              (Forall (Leq g k1) a)
              (Forall (Leq g' k2) a')
              TYPE (Whd FORALL)
              (check gamma Mix g g' k1)
              (check (citem : gamma) q c c' TYPE)
where
  citem = (Cxtb s (Leq g k1))
  s     = (create_name gamma)
  c     = unbind a s
  c'    = unbind a' s

```

Weak head mode, handling of bounds and polarity $+$: (AL-BOUND $_+$). In this case the first constructor is an application with a variable in its head, the second constructor has not the same variable in its head, and the variable has a bound in the context. Then the check mode is called with the first constructor, where the variable in the head is replaced by its bound, and with the second, which remains unchanged.

```

whd_mod gamma Co wh1 wh2
| (maybe1  $\neq$  Nothing)  $\wedge$ 
  (maybe2  $\equiv$  Nothing  $\vee$ 
    fromJust maybe1  $\neq$  fromJust maybe2)
)  $\wedge$  (g  $\neq$  Nothing) = UnNode gamma Co wh1 wh2 k0 (Whd BOUND)
  (check gamma Co
    (changeFirstArg wh1 (fromJust g))
    wh2 k0)
| otherwise = whd_inf gamma Co wh1 wh2
where
  maybe1 = hasHeadVar wh1
  maybe2 = hasHeadVar wh2
  g      = lookup_bound (fromJust maybe1) gamma

```

Weak head mode, handling of bounds and polarity $-$: (AL-BOUND $_-$). Very similar to the previous case.

```

whd_mod gamma Contra wh1 wh2
| (maybe2  $\neq$  Nothing)  $\wedge$ 

```

```

(maybex1 ≡ Nothing ∨
 fromJust maybex1 ≠ fromJust maybex2
) ∧ (g ≠ Nothing) = UnNode gamma Contra wh1 wh2 k0 (Whd BOUND)
  (check gamma Contra wh1
   (changeFirstArg wh2 (fromJust g))
   k0)
| otherwise = whd_inf gamma Contra wh1 wh2
where
  maybex1 = hasHeadVar wh1
  maybex2 = hasHeadVar wh2
  g        = lookup_bound (fromJust maybex2) gamma

```

Weak head mode, neutral terms (AL-NE). If the two constructors are neutral the inference mode is called.

```

whd_mod gamma p n n' = whd_inf gamma p n n'
whd_inf gamma p n n' =
  if (isNeutral n) ∧ (isNeutral n') then
    tree (inf_mod gamma p n n')
  else tree Nil
  where
    tree = λt → UnNode gamma p n n' k0 (Whd NE) t

```

Check mode ($\Gamma \vdash_a F \leq^q F' \Leftarrow \kappa$). The function *check* receives as arguments a context, a polarity, two constructors and a kind and returns a derivation tree.

```
check :: PCxt → Pol → Constr → Constr → Kind → DerTree
```

Check mode, $q = \top$. Then the algorithm succeeds immediately.

```
check gamma Inv c1 c2 k = UnNode gamma Inv c1 c2 k (Check INV_R) Leaf
```

Check mode, kind directed, $\kappa = p\kappa_1 \rightarrow \kappa_2$: (AL-FUN). A new variable of kind κ_1 is added to the context and the check mode is called recursively with the constructors applied to the new variable.

```

check gamma pol c1 c2 (ARR k1 k2) =
  UnNode gamma pol c1 c2 (ARR k1 k2) (Check FUN)
  (check (citem : gamma) pol (App c1 (Var a)) (App c2 (Var a)) k2)
where
  citem = (Cxt a k1)
  a     = (create_name gamma)

```

Check mode, kind directed, $\kappa = *$: (AL-WEAK). The constructors are weak head normalized and the weak head mode is called with them.

```

check gamma pol c1 c2 TYPE =
  UnNode gamma pol c1 c2 TYPE (Check WEAK)
  (whd_mod gamma pol w1 w2)

```

where

$$w1 = (whd\ c1)$$

$$w2 = (whd\ c2)$$

Bibliography

- [Abe06a] Andreas Abel. Implementing a normalizer using sized heterogeneous types. In Connor McBride and Tarmo Uustalu, editors, *Workshop on Mathematically Structured Functional Programming, MSFP 2006, Kuressaare, Estonia, July 2, 2006*, electronic Workshop in Computing (eWiC). The British Computer Society (BCS), 2006.
- [Abe06b] Andreas Abel. Polarized subtyping for sized types. In Dima Grigoriev, John Harrison, and Edward A. Hirsch, editors, *CSR*, volume 3967 of *Lecture Notes in Computer Science*, pages 381–392. Springer, 2006.
- [Abe06c] Andreas Abel. *A Polymorphic Lambda-Calculus with Sized Higher-Order Types*. PhD thesis, Ludwig-Maximilians-Universität München, 2006.
- [AG96] Ken Arnold and James Gosling. *The Java Programming Language*. The Java Series. Addison-Wesley, 1996.
- [Bac81] J. Backus. The history of FORTRAN I, II, and III. *ACM SIGPLAN Notices*, 18(6):25–74, June 1981.
- [BBC⁺97] Bruno Barras, Samuel Boutin, Cristina Cornes, Judicael Courant, Jean-Christophe Filliatre, Eduardo Gimenez, Hugo Herbelin, Gerard Huet, Cesar Munoz, Chetan Murthy, Catherine Parent, Christine Paulin-Mohring, Amokrane Saibi, and Benjamin Werner. The Coq proof assistant reference manual: Version 6.1. Technical Report RT-0203, INRIA, 1997.
- [BGR99] Paolo Baldan, Giorgio Ghelli, and Alessandra Raffaetà. Basic theory of F-bounded quantification. *Inf. Comput*, 153(1):173–237, 1999.
- [BSM⁺01] Gilad Bracha, David Stoutamire, Sun Microsystems, Philip Wadler, Bell Labs, and Lucent Technologies. GJ: Extending the java, December 18 2001.
- [CG99] Compagnoni and Goguen. Anti-symmetry of higher-order subtyping. In *CSL: 13th Workshop on Computer Science Logic*. LNCS, Springer-Verlag, 1999.
- [CG03] Adriana B. Compagnoni and Healfdene Goguen. Typed operational semantics for higher-order subtyping. *Information and Computation*, 184(2):242–297, 2003.
- [Chu40] A. Church. A simple theory of types. *Journal of Symbolic Logic*, 5:56–68, 1940.
- [CW85] Luca Cardelli and Peter Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, December 1985.
- [Gir71] J.-Y. Girard. Une extension de l’interprétation de Gödel à l’analyse, et son application à l’élimination des coupures dans l’analyse et la théorie des types. In

- J. E. Fenstad, editor, *2nd Scandinavian Logic Symposium*, pages 63–92. North-Holland, Amsterdam, 1971.
- [Gir72] Jean-Yves Girard. *Interprétation Fonctionnelle et Élimination des Compures de l'Arithmétique d'Ordre Supérieur*. Habilitation thesis, Université Paris VII, 1972.
- [Gog05] Healfdene Goguen. Justifying algorithms for $\beta\eta$ conversion. In Vladimiro Sassone, editor, *Proc. of the 8th Int. Conf. on Foundations of Software Science and Computational Structures, FoSSaCS 2005*, volume 3441 of *Lecture Notes in Computer Science*, pages 410–424. Springer-Verlag, 2005.
- [GR83] A. Goldberg and D. Robson. *Smalltalk 80: The Language and its Implementation*. Addison-Wesley, 1983.
- [Mey92] Bertrand Meyer. *Eiffel: The Language*. Prentice-Hall, 1992.
- [Pie92] Benjamin C. Pierce. Bounded quantification is undecidable. In *POPL*, pages 305–315, 1992.
- [Pie02] Benjamin C. Pierce. *Types and programming languages*. MIT Press, Cambridge, MA, USA, 2002.
- [Pol94] Robert Pollack. *The Theory of LEGO: A Proof Checker for the Extended Calculus of Constructions*. PhD thesis, University of Edinburgh, Edinburgh, Scotland, 1994.
- [PS97] Benjamin C. Pierce and Martin Steffen. Higher order subtyping. *Theoretical Computer Science*, 176(1,2):235–282, 1997.
- [Ste98] Martin Steffen. *Polarized Higher-Order Subtyping*. PhD thesis, Universität Erlangen-Nürnberg, 1998.
- [Str86] B. Stroustrup. *The C++ Programming Language*. Addison-Wesley, 1986.
- [WCPW03] Kevin Watkins, Iliano Cervesato, Frank Pfenning, and David Walker. A concurrent logical framework I:, July 29 2003.