Basic theory of F-bounded quantification

Paolo BALDAN, Giorgio GHELLI, Alessandra RAFFAETÀ
Dipartimento di Informatica,
Corso Italia, 40, 56125 Pisa, Italy
E-mail:{baldan, ghelli, raffaeta}@di.unipi.it

Proposed Running head:

F-bounded quantification

Corresponding author:

Giorgio Ghelli
Dipartimento di Informatica,
Corso Italia, 40,
I-56125 Pisa, Italy
Phone: ++39 050 887258
Fax: ++39 050 887226
E-mail: ghelli@di.unipi.it

List of Tables

1 A pictorial representation of rule II.5: the two subtyping derivations $(\forall\alpha \leq c_1.\, d_1); (\forall\alpha \leq c_2.\, d_2)$ and $\forall\alpha \leq (c_1[\alpha \leq A' \leftarrow c_2]).\, (d_1[\alpha \leq A' \leftarrow c_2]; d_2)$. 21
2 The equational system. 59

Abstract

System *F-bounded* is a second order typed lambda calculus, where the basic features of object-oriented languages can be naturally modelled. *F-bounded* extends the better known system $F_{\leq}$, in a way that provides an immediate solution for the treatment of the so-called "binary methods". Although more powerful than $F_{\leq}$ and also quite natural, system *F-bounded* has only been superficially studied from a foundational perspective and many of its essential properties have been conjectured but never proved in the literature. The aim of this paper is to give a solid foundation to *F-bounded*, by addressing and proving the key properties of the system. In particular transitivity elimination, completeness of the type checking semi-algorithm, the subject reduction property for $\beta\eta$ reduction, conservativity with respect to system $F_{\leq}$ and antisymmetry of a "full" subsystem are considered, and various possible formulations for system *F-bounded* are compared. Finally a semantic interpretation of system *F-bounded* is presented, based on partial equivalence relations.

1 Introduction

System *F-bounded* is a type system which was defined by Canning, Cook, Hill, Olthoff, and Mitchell [CCH$^+$89] to model the basic features of object-oriented languages. *F-bounded* properly generalizes system $F_{\leq}$ [CW85, CG92, CMMS94], an extension of the polymorphic lambda calculus (système F, [Rey74, Gir72]) with subtyping.

The key ingredient of $F_{\leq}$ is the bounded type abstraction (or *bounded polymorphism*). It allows one to define a function which works for every type A' that is a subtype of a bound A, and whose result type depends on A'. In this way $F_{\leq}$ integrates the expressive power of parametric polymorphism with that of subtyping, and it allows one to model many features of object-oriented languages, including subtyping and inheritance. However, an important class of object types, discussed later in Section 2, can be modelled naturally if the bound A of a quantified type variable α is allowed to depend on α itself, a situation forbidden in $F_{\leq}$. System *F-bounded* generalizes system $F_{\leq}$ by permitting this more liberal kind of quantification.

The essential properties of system $F_{\leq}$ have been extensively studied. One of the most important is transitivity elimination, i.e., the existence of an equivalent type system which is syntax driven and hence which does not contain an explicit transitivity rule [CG92]; a correct and complete type checking semi-algorithm is defined in the same paper. Another basic result is the undecidability of the subtype checking problem [Pie94], which immediately implies the undecidability of type checking. Termination of $\beta\eta$ reduction, namely the fact that every reduction sequence is finite, has been proved in [Ghe90]. Although $\beta\eta$ reduction in itself is not confluent, the confluence of reduction can be regained by adding a *Top* rule, which equates all terms with type *Top*. The $\beta\eta$*Top* system is normalizing (every term has a normal form), but it is still unknown whether it is terminating as well [CG94]. Finally, it has been proved that the extension of system $F_{\leq}$ with recursive types is not conservative, namely that, once recursive types are added, the traditional subtype checking algorithm is no longer complete, even with respect to non recursive types [Ghe93].

On the other hand, there have been few formal studies for system *F-bounded*. It was explicitly formalized for the first time in [Ghe97], where $\beta\eta$ reduction is proved to be terminating. The same property has been shown to hold for a Curry version (i.e., with implicitly typed variables) of the system, in [MKO95]. To the best of our knowledge, nothing else has been explicitly proved about system *F-bounded*. The lack of formal studies about system *F-bounded* is probably due to the fact that its similarity to system $F_{\leq}$ suggests that the two systems should enjoy the same properties. However this assumption must be carefully verified, since in many cases common beliefs on systems of the $F_{\leq}$ family have been discovered to be false. Moreover, system *F-bounded* differs from system $F_{\leq}$ at least in terms of the former having a subtyping relation which is not antisymmetric, and also in the behaviour of the standard subtype checking algorithm which is quite different in the two systems.[1]

For these reasons, we decided to try and prove some of the unproved key properties of system *F-bounded*, namely transitivity elimination, completeness of the

[1] Informally, the class of judgements which make the standard subtype checker diverge is significantly larger in system *F-bounded*.

type checking semi-algorithm, subject reduction for $\beta\eta$ reduction, undecidability of subtype checking, conservativity with respect to system $F_{\leq}$, non conservativity of strong recursive types, definition of an antisymmetric subsystem, and the equivalence (under suitable conditions) of the two different versions which have been proposed in the literature for the key subtyping rule for bounded quantification. Although the results are not surprising, we believe that it was time to prove them in order to give a solid foundation for further studies about system *F-bounded*.

Another contribution of this paper is the presentation of an explicit semantic interpretation of system *F-bounded*. Our interpretation is a classical realizability interpretation based on partial equivalence relations, in the tradition of [BL90, CL91, CMMS94, Ghe90] and others. However, most of these papers focus on "semantic frameworks", i.e., on the definition of a general notion of a semantic interpretation for system $F_{\leq}$, in the style of [BMM90], rather than on the definition of a specific interpretation. As a consequence, most proofs become quite complex, and are not actually reported, but the reader is referred to a chain of classical papers. Here we address just one specific interpretation, and although the techniques we use are standard, we give explicit proofs of the key properties, though we leave it to the reader to generalize the interpretation. We think that this is an interesting complement to what is obtained in the more general framework-based approach.

The rest of the paper is structured as follows. Section 2 discusses how system *F-bounded* may be used to model some basic features of object-oriented languages. Section 3 presents a formal definition of system *F-bounded*. Section 4 studies the subtype relation in system *F-bounded* and in particular proves the transitivity elimination property. Section 5 introduces a type checking semi-algorithm for system *F-bounded*, and proves its correctness and completeness. Section 6, relying on transitivity elimination, proves the subject reduction property for *F-bounded* with β and η reduction rules. Section 7 studies type equivalence for system *F-bounded* and shows that, although subtyping is not antisymmetric in this system, two different types are equivalent if, and only if, one can be transformed into the other by changing $\alpha \leq \alpha$ into $\alpha \leq \mathit{Top}$ bounds. This result naturally suggests how an antisymmetric equivalent subsystem can be defined. Section 8 characterizes the relationship between our version of system *F-bounded* and some other less expressive variants. Section 9 proves that system *F-bounded* is conservative with respect to system $F_{\leq}$; as a corollary, this immediately implies that subtyping for system *F-bounded* is undecidable, and that the addition of recursive types gives a non-conservative extension of *F-bounded*. Section 10 defines a semantic interpretation for system *F-bounded*, along the now classical lines of [BL90]. Finally, Section 11 draws some conclusions.

2 System *F-bounded* and object-oriented programming

System *F-bounded* has been proposed as a foundation for the type system of object-oriented programming, since it offers all the basic mechanisms which are needed to define a rich object-oriented language. The fundamental features which should be offered by this kind of languages may be listed as follows.[2]

[2]Here we consider the class-based view of object-oriented languages. However also the alternative object-based view, which is slightly different, can be described in the context of system

- *Type abstraction*: the ability to define abstract data types (ADTs), consisting of a *name*, an *interface*, which lists the possible operations on values of that type (the *messages*), and an *implementation*, which defines a representation for the objects of that type and an implementation for the operations in the interface (the *methods* of the messages). Values of an ADT can only be manipulated through the operations offered by the interface. Type abstraction is not unique to object-oriented languages, but it is their most important feature; in this context, an ADT is called a *class*, and any value of the class is represented as a record.

- *Inheritance*: the ability to define the interface and the implementation of a class by saying how it differs from a "superclass". A definition by inheritance of an interface can either add new messages to the superclass interface or change their type. A definition by inheritance of an implementation can either add new fields to the record type used to represent the class values, or add methods to the superclass, or change the method of a message (method *overriding*).

- *Subtyping*: a (pre)order relation among types such that, if a type T is a subtype of U, every function which is defined on U can also operate on values of type T. Usually inheritance is linked to subtyping, i.e., the type defined by a subclass is a subtype of the type defined by the superclass.

- *Overloading with late binding*: suppose that both a type U and a subtype T of U have a message m in their interface, but they define two different methods M_U and M_T for that message. Then the application $o.m$ of the message m to an object o may either invoke M_U or M_T, depending on the type of o; we say that the message m is *overloaded*. Due to subtyping, the type inferred for o by the compiler is only a supertype of the type of the values which can be denoted by o. For example, if o is the formal parameter of type U of a function, the compiler gives o the type U, but in different invocations of the function it may be bound to values of type U or to values of the subtype T. In this case, a language with an early binding (or *static binding*) resolution mechanism translates (at compile time) the application $o.m$ to a call to the method M_U. Instead, a language with a late binding (or *dynamic binding*) resolution mechanism will translate (at run time) the application $o.m$ with the method which is the most appropriate for the actual parameter of the function.

- *Late binding of* self: every method can send messages to a special variable, often called *self*. This variable denotes the object which has received the message whose method is executing; messages sent to *self* are resolved using late binding.

The fact that ADTs could be encoded in system F is well known, and was first studied in [MP88].[3] Subtyping can be added to system F in many different ways, the most widely accepted being the one proposed by Cardelli and Wegner

F-bounded (see [AC96b]).

[3] Object-oriented ADT's are actually best understood as a form of *procedural abstraction*, according to the distinction introduced by Reynolds [Rey74, Coo91]; this kind of abstraction can be still represented in system F.

in [CW85] and formalized as system $F_{\leq}$ in [CG92, CMMS94]. As mentioned in the introduction, the key feature of system $F_{\leq}$ is the bounded type abstraction, which allows one to define a function that works for every type A' which is a subtype of a type bound A, producing a result whose type depends on A'. System $F_{\leq}$, enriched with recursive values and types, is expressive enough to allow one to encode most key constructs of object-oriented languages, such as the inheritance of implementations, late binding and self variables. However, this approach does not deal smoothly with *binary methods*. A binary method is a method that takes a parameter of the same type as the receiving object, as in the canonical example of an object type *Point* with a binary method which tests for equality. In this example we assume that an object type only specifies the interface of methods, and not their implementation; the operator $Rec\ X.T$ defines a recursive type.

```
Point = Rec X.
  [ x: Int;
    eq: X -> Bool
  ]
```

Consider now a new object type *ColouredPoint*, which adds a method *colour* to the type *Point*, as follows.

```
ColouredPoint = Rec X.
  [ x: Int;
    eq: X -> Bool;
    colour: Colour
  ]
```

A type B which is defined by adding some fields to an object type A is said to *match* A. The basic observation is that when A has some binary methods, B may match A without being a subtype of A. For example, the type *ColouredPoint* is not a subtype of *Point* since the type $ColouredPoint \rightarrow Bool$ of the *eq* field of *ColouredPoint* is not a subtype of the type $Point \rightarrow Bool$ of the *eq* field of *Point*. The absence of subtyping is also witnessed by the fact that a *ColouredPoint* cannot appear in any context where a *Point* can appear. For example, an expression *x.eq(y)* is type correct when both x and y are of type *Point*, but it may raise an exception if x is substituted with an object of type *ColouredPoint*: its equality function expects a *ColouredPoint* parameter, hence it may try and access the *colour* field of y.

On the other hand, although *ColouredPoint* is not a subtype of *Point*, most functions that operate on points may correctly operate on coloured points and on any other type which matches the *Point* type, but no type for these functions can be expressed in system $F_{\leq}$. By permitting the presence of a type variable in its own bound, *F-bounded* quantification allows the programmer to express the fact that a function works with any type that matches the *Point* type, by assigning to such function a type:

$$\forall \alpha \leq [x : Int; eq : \alpha \rightarrow Bool].\, \alpha \rightarrow B$$

The condition $\alpha \leq [x : Int; eq : \alpha \rightarrow Bool]$ is satisfied by any type which is obtained by adding some fields to the recursive definition of type *Point* or by

specializing some of the non-recursive fields, and even by some other types.[4] This form of quantification allows one to write many useful functions which operate on all the types which match the *Point* type.

F-bounded quantification is not the only way to give this kind of functions a type. It is also possible to define a "matching" relation which is different from the subtype relation, and to quantify functions on every type that matches an object type A [Bru94, BSvG95]; however, this approach is slightly more complex and ad hoc than the *F-bounded* one (see [BCC$^+$96]). Another possibility arises when one considers higher order type systems, such as $F^\omega_\leq$, where it is possible to define type operators, i.e., functions from types to types [Car90, PS97]. In this context a quantification:

$$\Lambda\alpha \leq A[\alpha].\, b[\alpha]$$

can be expressed as:

$$\Lambda\alpha \leq (\Lambda_2\beta.A[\beta]).\, b[Fix(\alpha)],$$

where Λ represents abstraction of terms over types, Λ_2 represents abstraction of types over types, and *Fix* is a fixpoint operator (see [AC96a] for details). This approach is very interesting, but the recursive version of system $F^\omega_\leq$ has not been completely understood yet. In particular, strong recursive types (the notion of *strong recursion* is discussed in Sections 9 and 11) have not been studied in this context, and it is not yet known how to combine type operators with the full $F_\leq$ subtype system.[5] Hence, we have no hope of deriving properties of system *F-bounded* from such an encoding.

Another way to deal with binary methods is to switch from seeing objects as records which contain their methods, to seeing a method as an overloaded function. This complementary approach has some advantages, above all that methods become first class values and that it is possible to deal smoothly both with contravariant and covariant overriding of methods, and in particular with binary methods. This approach was first proposed in [Ghe91], in the context of strongly typed languages, and then studied in [CGL95, CGL93, Cas96, Cas97]. Though very promising, this approach has not been studied sufficiently, and there are some problems in the definition of a clean semantic model and in the design of a suitable type abstraction mechanism to bind the definition of methods with the definition of the corresponding class.

For more information and references about the problem of binary methods we refer the reader to the excellent paper [BCC$^+$96].

To conclude this section it is worth remarking that for practical purposes *F-bounded* quantification alone is not very useful, because system *F-bounded*, as studied in this paper, should be enriched with a notion of recursive types in order to model an object-oriented language with binary methods. We concentrate, however, on the core system, with no recursion at the value or at the type level, in order to understand the basic properties of *F-bounded* quantification. Only some suggestions are given on how the system properties would be affected by the addition of recursive types (see Sections 9 and 11). While the core system is rich enough to

[4] For example, a version of the *ColouredPoint* type whose *eq* field has type $Point \to Bool$ would satisfy the type inequality considered.

[5] So far, only the kernel-fun version (Section 11) of system $F^\omega_\leq$ has been studied in detail.

merit studying, this work is also intended to provide a foundation for future studies of extensions of *F-bounded* with recursion.

3 System *F-bounded*

This section defines system *F-bounded* by formalizing the intuitive ideas presented above. The starting point is the second order typed lambda calculus, where besides value abstraction (λ), a second order feature of type abstraction (Λ) is present. Subtyping and the possibility to specify a bound for a quantified type variable are added by system $F_{\leq}$. *F-bounded* is obtained from $F_{\leq}$ by relaxing the constraint which disallows the presence of a type variable in its bound.

3.1 Syntax for Types, Terms, Environments and Judgements

Many different formulations for system *F-bounded* are possible. First of all, one can adopt either an explicit approach (à la Church) where every variable is annotated with its type, and where type abstractions and applications are explicit, or an implicit approach (à la Curry), where types are inferred rather than appearing inside terms. In line with tradition, we will adopt the explicit approach, which gives programmers a finer control over the typing of the terms they write. There are two minor syntactic variants to be considered:

1. in a type good formation judgement $\Gamma \vdash A$, the environment Γ may contain just a list of variable names (like in [Ghe97], or in presentations of system F) or it may contain a list of variables with their bounds (like in [CG92]);

2. in a typing judgement $\Gamma \vdash a : A$, typing and subtyping assumptions may be mixed in Γ (like in [CG92, CMMS94]) or separated (like in [Ghe97]).

Both choices are mainly stylistic and have minor advantages; we opted for the second possibility in both cases. Finally, we may allow or forbid a type variable to be a bound for itself (as in $\forall\alpha\leq\alpha.\, A$). We will allow this kind of bound, while the variant where this is forbidden will be studied in Section 8.

Let *TypeVar* and *ValVar* be two fixed countable (disjoint) sets of variables referred to as *type variables* and *value variables* respectively. The set *TypeVar* will be ranged over by Greek letters α, β, ..., while *ValVar* will be ranged over by Latin letters x, y, ... The syntax of our system is then described by the following grammar.

$$
\begin{array}{lcll}
A & ::= & \alpha \mid \mathit{Top} \mid A \to A \mid \forall\alpha\leq A.\, A & \text{(Types)} \\
a & ::= & x \mid \lambda x{:}A.\, a \mid a(a) \mid \Lambda\alpha\leq A.\, a \mid a\{A\} & \text{(Terms)} \\
\Gamma & ::= & \epsilon \mid \Gamma, \alpha\leq A & \text{(TypeEnv)} \\
\Delta & ::= & \epsilon \mid \Delta, x{:}A & \text{(ValueEnv)} \\
J & ::= & \Gamma \vdash \diamondsuit \mid \Gamma \vdash A \mid \Gamma, \Delta \vdash \diamondsuit \mid \Gamma \vdash A \leq A \mid \Gamma, \Delta \vdash a : A & \text{(Judgements)}
\end{array}
$$

The *arrow type* $A \to B$ is the type of functions taking arguments of type A and giving back results of type B. The *bounded universal type* $\forall\alpha \leq A.\, B$ is the

type of terms which, when applied to a type A', yield a term of type $B[\alpha \leftarrow A']$; the application is allowed only when A' is a subtype of $A[\alpha \leftarrow A']$. The *Top* type is a supertype of every type. In system $F_{\leq}$, the *Top* type gives the system the expressive power needed to encode records and objects; system $F_{\leq}$ without the type *Top* would be decidable, but it would be impossible to encode record types (see [CMMS94], [KS92]). The role of this type in system *F-bounded* will be discussed in Section 8.

Among terms, we find the three forms of the classical typed lambda calculus, plus *type abstraction* $\Lambda\alpha \leq A.\, a$ and *type application* $a\{A\}$, whose use has been informally exemplified in the previous section.

Two different kinds of environments are present. A *type environment* Γ consists of a list of type variables, each bounded by a type bound. A *value environment* Δ consists of a list of value variables, each one bound to a type.

Finally, judgements represent the assertions we can express about our calculus. The judgement $\Gamma \vdash \diamondsuit$ means that Γ is a well-formed type environment, i.e., that no type variable occurs free in Γ. More precisely, Γ is well-formed if every free variable in the bound A of a variable α is either α or it is defined in the part of Γ on the left of $\alpha \leq A$. The judgement $\Gamma \vdash A$ means that the type A is well-formed in the environment Γ, i.e., that Γ is well-formed and every free variable in A is defined in Γ. The judgement $\Gamma, \Delta \vdash \diamondsuit$ means that Γ is a well-formed type environment, and that the value environment Δ is well-formed in Γ, i.e., that the type of each variable in Δ is well-formed in Γ. The judgement $\Gamma \vdash A \leq A'$ means that A is a subtype of A', and they are both well-formed types in Γ. Lastly, $\Gamma, \Delta \vdash a : A$ means that the term a has type A when the assumptions in Γ, Δ hold. The environment Γ must be a well-formed type environment, and Δ and A well-formed in Γ.

3.2 The rules

We are now ready to introduce the good formation, subtyping and typing rules for system *F-bounded*. Before getting into technical definitions we will clarify our conventions in the treatment of (type and value) variables. We adopt the De Bruijn approach [dB72], where variables are not considered as names but as pointers to the surrounding context (free variables are then simply pointers that "go outside the context"). However, working explicitly with De Bruijn indexes, notation becomes cumbersome and the readability of terms decreases considerably. Therefore we will continue working with variable names, but simply as a more convenient way of denoting De Bruijn pointers. The advantage of this approach is that no α-conversion is needed, since each α-congruent class of ordinary terms corresponds exactly to one nameless De Bruijn term. To have some more details on the actual De Bruijn definition please refer to the appendix.

First of all we define the set of free variables for types, value environments and terms, and we give the corresponding formation rules, formalizing the intuition given in the previous subsection.

Free type variables for:

- **Types**

$$FV(\alpha) = \{\alpha\} \qquad\qquad FV(A \rightarrow B) = FV(A) \cup FV(B)$$

$$FV(Top) = \emptyset \qquad FV(\forall \alpha \leq A.\, B) = (FV(A) \cup FV(B)) \setminus \{\alpha\}$$

- **Value Environments**

$$FV(\epsilon) = \emptyset \qquad FV(\Delta, x{:}A) = FV(\Delta) \cup FV(A)$$

The same symbol FV will also be used to denote the set of free (type and value) variables in a term, defined as follows:

$$FV(x) = \{x\}$$

$$FV(\lambda x{:}A.\, b) = FV(A) \cup (FV(b) \setminus \{x\}) \qquad FV(f(a)) = FV(f) \cup FV(a)$$

$$FV(\Lambda \alpha \leq A.\, b) = (FV(A) \cup FV(b)) \setminus \{\alpha\} \qquad FV(b\{A\}) = FV(b) \cup FV(A)$$

Given a type environment $\Gamma \equiv \alpha_1 \leq A_1, \ldots, \alpha_n \leq A_n$ we denote with $vars(\Gamma)$ the set of type variables bounded in Γ, i.e., $\{\alpha_1, \ldots, \alpha_n\}$. Moreover, we denote with $\Gamma(\alpha_i)$ the type A_i for $i \in \{1, \ldots, n\}$; in the De Bruijn notation, every free variable in A_i has to be adjusted so that it points to the same binder as before (technically, in the judgement $\Gamma \vdash \alpha \leq \Gamma(\alpha)$, the offset $n - i$ has to be added to the index of every free variable in A_i). In the same way, given a value environment $\Delta \equiv x_1{:}A_1, \ldots, x_n{:}A_n$ we denote with $vars(\Delta)$ the set of value variables typed in Δ, i.e., $\{x_1, \ldots, x_n\}$ and with $\Delta(x_i)$ the type A_i.

Type environment formation rules

$$\epsilon \vdash \diamondsuit \quad (\epsilon\mathsf{TEnv}) \qquad \frac{\Gamma \vdash \diamondsuit \quad FV(A) \subseteq vars(\Gamma) \cup \{\alpha\}}{\Gamma, \alpha \leq A \vdash \diamondsuit}\ (\mathsf{TEnv})$$

Type formation rules

$$\frac{\Gamma \vdash \diamondsuit \quad FV(A) \subseteq vars(\Gamma)}{\Gamma \vdash A}\ (\mathsf{TypeForm})$$

Value environment formation rules

$$\frac{\Gamma \vdash \diamondsuit}{\Gamma, \epsilon \vdash \diamondsuit}\ (\epsilon\mathsf{VEnv}) \qquad \frac{\Gamma, \Delta \vdash \diamondsuit \quad \Gamma \vdash A}{\Gamma, \Delta, x{:}A \vdash \diamondsuit}\ (\mathsf{VEnv})$$

Subtype rules

$$\frac{\Gamma \vdash A}{\Gamma \vdash A \leq A}\ (\mathsf{Id} \leq) \qquad \frac{\Gamma \vdash A \leq B \quad \Gamma \vdash B \leq C}{\Gamma \vdash A \leq C}\ (\mathsf{Trans} \leq)$$

$$\frac{\Gamma \vdash \diamondsuit \quad \alpha \in vars(\Gamma)}{\Gamma \vdash \alpha \leq \Gamma(\alpha)}\ (\mathsf{Var} \leq) \qquad \frac{\Gamma \vdash A}{\Gamma \vdash A \leq Top}\ (\mathsf{Top} \leq)$$

$$\frac{\Gamma \vdash A' \leq A \quad \Gamma \vdash B \leq B'}{\Gamma \vdash A \to B \leq A' \to B'} \ (\to\leq)$$

$$\frac{\Gamma, \alpha \leq A' \vdash \alpha \leq A \quad \Gamma, \alpha \leq A' \vdash B \leq B'}{\Gamma \vdash (\forall \alpha \leq A.\, B) \leq (\forall \alpha \leq A'.\, B')} \ (\forall \leq)$$

Term formation rules

$$\frac{\Gamma, \Delta \vdash \diamondsuit \quad x \in vars(\Delta)}{\Gamma, \Delta \vdash x : \Delta(x)} \ (\mathsf{Var}) \qquad \frac{\Gamma, \Delta \vdash a : A \quad \Gamma \vdash A \leq B}{\Gamma, \Delta \vdash a : B} \ (\mathsf{Subs})$$

$$\frac{\Gamma, \Delta, x{:}A \vdash b : B}{\Gamma, \Delta \vdash \lambda x{:}A.\, b : A \to B} \ (\to \mathsf{I}) \qquad \frac{\Gamma, \Delta \vdash f : A \to B \quad \Gamma, \Delta \vdash a : A}{\Gamma, \Delta \vdash f(a) : B} \ (\to \mathsf{E})$$

$$\frac{\Gamma, \alpha \leq A, \Delta \vdash b : B \quad \alpha \notin FV(\Delta)}{\Gamma, \Delta \vdash (\Lambda \alpha \leq A.\, b) : (\forall \alpha \leq A.\, B)} \ (\forall \mathsf{I})$$

$$\frac{\Gamma, \Delta \vdash f : \forall \alpha \leq A.\, B \quad \Gamma \vdash A' \leq A[\alpha \leftarrow A']}{\Gamma, \Delta \vdash f\{A'\} : B[\alpha \leftarrow A']} \ (\forall \mathsf{E})$$

Notation 3.1 When necessary to avoid ambiguity, a judgement derivable in *F-bounded* will be denoted as

$$Pre \vdash_{\flat} Concl$$

Notice that the fact that *F-bounded* is a proper extension of $F_{\leq}$ is essentially expressed by the type environment formation rule (TEnv), which allows the type variable α to occur free in its bound A. Indeed, although some other *F-bounded* rules are slightly different from the corresponding $F_{\leq}$ rules, it can be seen that (a system equivalent to) $F_{\leq}$ can be simply regained by strengthening the second premise of rule (TEnv) into $FV(A) \subseteq vars(\Gamma)$. The relation between $F_{\leq}$ and *F-bounded* will be studied in Section 9.

We finally present the reduction rules of the system. Notice that there are two kinds of β and η rules: besides the usual rules of the (typed) lambda calculus (βTerm) and (ηTerm), the corresponding rules regarding type abstraction are present. The four rules define a binary relation that, closed by context, gives the one-step reduction relation, and then, closed by reflexivity and transitivity, gives the many-steps reduction relation.

Reduction rules

$$(\lambda x{:}A.\, b)(a) \twoheadrightarrow b[x \leftarrow a] \qquad (\beta\mathsf{Term})$$

$$(\Lambda \alpha \leq A.\, b)\{A'\} \twoheadrightarrow b[\alpha \leftarrow A'] \qquad (\beta\mathsf{Type})$$

$$\lambda x{:}A.\, b(x) \twoheadrightarrow b \quad \text{if } x \notin FV(b) \qquad (\eta\mathsf{Term})$$

$$\Lambda \alpha \leq A.\, b\{\alpha\} \twoheadrightarrow b \quad \text{if } \alpha \notin FV(b) \qquad (\eta\mathsf{Type})^{6}$$

[6] Using the De Bruijn notation, in rule (ηTerm), the b at the right hand side is obtained by decrementing every free variable in the b of the left hand side by one, so that every variable still points to the same binder. The same consideration applies to rule (ηType).

This relation is terminating [Ghe97], but not confluent, due to subtyping. Consider the two following normalizing reductions.

$$\lambda x{:}A.\,(\lambda y{:}B.\,y)x \twoheadrightarrow_{\eta} \lambda y{:}B.\,y$$

$$\lambda x{:}A.\,(\lambda y{:}B.\,y)x \twoheadrightarrow_{\beta} \lambda x{:}A.\,x$$

Since the term $\lambda x : A.\,(\lambda y : B.\,y)x$ is well typed for any $A \leq B$, the above critical pair is not confluent in any calculus with a non-trivial subtype relation. This problem has been addressed in system $F_{\leq}$ by proving that $\beta\eta$ reduction can be made confluent by adding a *Top* rule, which equates every term with a *Top* type, plus some rules which may be obtained by a Knuth-Bendix like process [CG94]. The same approach may also apply to system *F-bounded*, but we leave this as an open issue.

4 Transitivity elimination

Transitivity plays a central role in every subtype system. In fact, requiring the transitivity of the subtype relation is fundamental both from a conceptual and from a technical point of view. First of all, the informal understanding of subtyping is based on the notion of set inclusion: "integer" is a subtype of "real" since every integer number is also a real number. Moreover, subtyping formalizes the idea of "specialization of properties" in the following sense: T is a subtype of U if every relevant property of all values of type U is also enjoyed by all values of type T. Clearly both the set inclusion and the "specialization" relations are transitive. More technically, we will see that transitivity is very important to prove the subject reduction property.

However, the presence of an explicit rule for transitivity makes it difficult to decide the subtyping relation. The standard subtype checking algorithm takes a couple of types and an environment, and searches for a rule whose conclusion matches the judgement to be proved. If no matching rule is found, then the judgement cannot be proved. If only one matching rule is found, then the problem can be reduced to the problem of proving all the premises of the rule. If many matching rules exist, then each one of them must be tried, in a non deterministic fashion.

This algorithm cannot be applied in the presence of a transitivity rule. First of all, every subtyping judgement matches its conclusion, hence the algorithm will never give a negative answer. Moreover, both premises contain a metavariable (the type B, in our formulation in Section 3) which is not instantiated by the conclusion, and whose value must hence be guessed by the algorithm.

To solve this problem, it is customary to define two different presentations for a subtype system: an *abstract* presentation and an *algorithmic* one. The *abstract* presentation contains one subtyping rule for every form of type, and a transitivity rule which ensures the transitivity of the whole system. This presentation is aimed at describing the system in the most understandable way. On the other hand, the *algorithmic* presentation is defined in order to allow for a direct application of the standard algorithm. To this aim, the rules are modified in such a way that no judgement may match the conclusion of two different rules. Moreover every variable in the premises of a rule also appears in its conclusion, and thus no guessing is needed; in particular, the full transitivity rule is not inserted in the algorithmic

presentation (but it may be embedded in some other rules, as happens with rule ($\mathsf{TVar} \leq$) below). Finally, one proves that the two sets of rules define the same relation. This proof is called *proof of transitivity elimination*, since it shows that the abstract presentation can be transformed into an equivalent presentation with no transitivity rule.

4.1 The algorithmic presentation

In our case, the algorithmic presentation is obtained from the original system by removing the transitivity rule ($\mathsf{Trans} \leq$). Moreover the rule for variables ($\mathsf{Var} \leq$) is replaced by a new rule containing a "restricted form" of transitivity and the identity rule ($\mathsf{Id} \leq$) is specialized to work only on type variables.

Definition 4.1 ***(deterministic* F-bounded*)*** The system *dF-bounded* (*deterministic F-bounded*) is obtained from *F-bounded* by removing the rule ($\mathsf{Trans} \leq$) and by substituting the rules ($\mathsf{Var} \leq$) and ($\mathsf{Id} \leq$) with the following ones:

$$\frac{\Gamma \vdash \Gamma(\alpha) \leq A \quad \alpha \in \mathit{vars}(\Gamma) \quad \Gamma(\alpha) \neq \alpha, \mathit{Top} \quad A \not\equiv \alpha, \mathit{Top}}{\Gamma \vdash \alpha \leq A} \ (\mathsf{TVar} \leq)$$

$$\frac{\Gamma \vdash \Diamond \quad \alpha \in \mathit{vars}(\Gamma)}{\Gamma \vdash \alpha \leq \alpha} \ (\mathsf{IdVar} \leq)$$

Hereafter the premise $\alpha \in \mathit{vars}(\Gamma)$ of rule ($\mathsf{TVar} \leq$) will be often omitted, since we consider it to be implied by the use of the notation $\Gamma(\alpha)$ in the other premises.

Notice that the set of subtyping rules of *dF-bounded* is deterministic (or syntax-directed), in the sense that given any subtyping judgement $\Gamma \vdash A \leq B$ there is at most one rule that can be applied to obtain that conclusion. Therefore, as anticipated, the standard algorithm which, given a judgement, tries to construct a proof of that judgement in *dF-bounded*, is deterministic (no backtracking is needed).

4.2 The proof of transitivity elimination

The proof of transitivity elimination is based on the introduction of an intermediate system, called *F-bounded*$^+$, equivalent to *F-bounded*. *F-bounded*$^+$ is then proved to be equivalent also to the algorithmic system *dF-bounded*, by showing that a suitable set of rewrite rules allows us to reduce each *F-bounded*$^+$ derivation into a normal form derivation with the same conclusion, which turns out to be a *dF-bounded* derivation. Then transitivity elimination immediately follows.

4.2.1 System *F-bounded*$^+$

The system *F-bounded*$^+$ is obtained from *F-bounded* by replacing the ($\mathsf{Var} \leq$) subtyping rule with the rule ($\mathsf{TVar} \leq$) of system *dF-bounded*.

System *F-bounded*$^+$ is clearly equivalent to *F-bounded*: given any *F-bounded* proof we can obtain an *F-bounded*$^+$ proof with the same premises and conclusion by replacing every instance of the rule ($\mathsf{Var} \leq$) with a subproof combining the rules ($\mathsf{TVar} \leq$) and ($\mathsf{Id} \leq$); in the other direction, every instance of rule ($\mathsf{TVar} \leq$) can be substituted by an instance of ($\mathsf{Var} \leq$) plus transitivity. It is worth noticing that *dF-bounded* can be obtained from *F-bounded*$^+$ by removing rule ($\mathsf{Trans} \leq$) and restricting the use of ($\mathsf{Id} \leq$) to type variables.

Notation 4.2 When necessary to avoid ambiguity, judgements derivable in *F-bounded*$^+$ and in *dF-bounded* will be denoted respectively as

$$Pre \vdash_+ Concl \qquad \text{and} \qquad Pre \vdash_d Concl$$

In *F-bounded*$^+$, it is convenient to have a linear notation for subtyping derivations, so that operations performed on derivations to reduce them to normal form can be expressed as textual rules (see [CG92, Pie97]). By $c :: J$ we mean that c is a derivation whose conclusion is the judgement J. The same notation will be used to indicate that c is the linear abbreviation of a derivation having J as its conclusion, i.e., when the meaning is clear from the context, we identify a derivation with its linear representation.

Definition 4.3 ***(linear abbreviations for derivations)*** The translation function $(\cdot)^\dagger$, which maps derivation trees (in *F-bounded*$^+$) to their abbreviated forms, is defined by induction on the structure of the derivation:

$$\left(\frac{\Gamma \vdash A}{\Gamma \vdash_+ A \leq A}\ (\mathsf{Id}\leq) \right)^\dagger = Id_{\Gamma,A}$$

$$\left(\frac{c :: \Gamma \vdash_+ \Gamma(\alpha) \leq A \quad \Gamma(\alpha) \neq \alpha, Top \quad A \not\equiv \alpha, Top}{\Gamma \vdash_+ \alpha \leq A}\ (\mathsf{TVar}\leq) \right)^\dagger = V_{\alpha,A}(c^\dagger)$$

$$\left(\frac{\Gamma \vdash A}{\Gamma \vdash_+ A \leq Top}\ (\mathsf{Top}\leq) \right)^\dagger = Top_{\Gamma,A}$$

$$\left(\frac{c :: \Gamma \vdash_+ A' \leq A \quad d :: \Gamma \vdash_+ B \leq B'}{\Gamma \vdash_+ A \to B \leq A' \to B'}\ (\to\leq) \right)^\dagger = (c^\dagger \to d^\dagger)$$

$$\left(\frac{c :: \Gamma, \alpha \leq A' \vdash_+ \alpha \leq A \quad d :: \Gamma, \alpha \leq A' \vdash_+ B \leq B'}{\Gamma \vdash_+ \forall \alpha \leq A.\, B \leq \forall \alpha \leq A'.\, B'}\ (\forall\leq) \right)^\dagger = (\forall \alpha \leq c^\dagger.\, d^\dagger)$$

$$\left(\frac{c :: \Gamma \vdash_+ A \leq B \quad d :: \Gamma \vdash_+ B \leq C}{\Gamma \vdash_+ A \leq C}\ (\mathsf{Trans}\leq) \right)^\dagger = (c^\dagger; d^\dagger)$$

Notice that only the abbreviations for the basic derivations using ($\mathsf{Id} \leq$) and ($\mathsf{Top} \leq$) are explicitly adorned with the environment since in the other cases the environment is already coded in the premises. Furthermore, sometimes we will not indicate explicitly the environment involved in a derivation unless it is strictly necessary. Therefore we will write Top_A for $Top_{\Gamma,A}$, and Id_A for $Id_{\Gamma,A}$.

4.2.2 Replacement, Top-lemmata and weakening

We collect here some definitions and technical lemmata which will be useful below. Although most of the lemmata are formulated for *F-bounded*$^+$, it is easy to verify that their obvious reformulations for the other systems considered so far hold as well. For this reason, we will sometimes refer and apply them to systems *F-bounded* or *dF-bounded* as well.

The first lemma specifies that (sub)typing implies good formation for the types and environment involved.

Lemma 4.4 (subproof) *If* $\Gamma \vdash A \leq B$ *then* $\Gamma \vdash \Diamond$, $\Gamma \vdash A$ *and* $\Gamma \vdash B$. *If* $\Gamma \vdash a : A$ *then* $\Gamma \vdash \Diamond$ *and* $\Gamma \vdash A$.

Proof. Routine induction on the structure of the derivations. □

Replacement is an operation on derivations, which allows us to replace a hypothesis $\alpha \leq A'$ in the environment $\Gamma, \alpha \leq A', \Gamma'$ of a derivation, with another hypothesis $\alpha \leq A$, whenever $\Gamma, \alpha \leq A, \Gamma'$ proves $\alpha \leq A'$.

Definition 4.5 ***(derivations replacement)*** Let $c :: \Gamma, \alpha \leq A, \Gamma' \vdash_+ \alpha \leq A'$ and $d :: \Gamma, \alpha \leq A', \Gamma' \vdash_+ B \leq B'$ be *F-bounded*$^+$ derivations. The *replacement* of $\alpha \leq A'$ in d with c, denoted by $d[\alpha \leq A' \leftarrow c]$, is defined by induction on the structure of d as follows:

1. $Id_{(\Gamma, \alpha \leq A', \Gamma'), B'}[\alpha \leq A' \leftarrow c] = Id_{(\Gamma, \alpha \leq A, \Gamma'), B'}$
2. $V_{\beta, B'}(d_1)[\alpha \leq A' \leftarrow c] = \begin{cases} V_{\beta, B'}(d_1[\alpha \leq A' \leftarrow c]) & \text{if } \beta \not\equiv \alpha \\ c; (d_1[\alpha \leq A' \leftarrow c]) & \text{if } \beta \equiv \alpha \end{cases}$

   (Observe that when $\beta \equiv \alpha$ one has $B \equiv \alpha$ and $d_1 :: \Gamma, \alpha \leq A', \Gamma' \vdash_+ A' \leq B'$.)
3. $Top_{(\Gamma, \alpha \leq A', \Gamma'), B'}[\alpha \leq A' \leftarrow c] = Top_{(\Gamma, \alpha \leq A, \Gamma'), B'}$
4. $(d_1 \to d_2)[\alpha \leq A' \leftarrow c] = ((d_1[\alpha \leq A' \leftarrow c]) \to (d_2[\alpha \leq A' \leftarrow c]))$
5. $(\forall \beta \leq d_1 . d_2)[\alpha \leq A' \leftarrow c] = (\forall \beta \leq (d_1[\alpha \leq A' \leftarrow c]) . (d_2[\alpha \leq A' \leftarrow c]))$
6. $(d_1; d_2)[\alpha \leq A' \leftarrow c] = ((d_1[\alpha \leq A' \leftarrow c]); (d_2[\alpha \leq A' \leftarrow c]))$

Notice that some parentheses in the definition are not genuine syntactical objects. They are inserted only for the sake of clarity.

The replacement operation has two main effects. First, it substitutes every instance of $\alpha \leq A'$ in the environment Γ of basic derivations with $\alpha \leq A$ (rules 1 and 3). Second, whenever $\Gamma, \alpha \leq A', \Gamma' \vdash \alpha \leq B'$ is proved by applying rule ($\mathsf{TVar} \leq$) to $\Gamma, \alpha \leq A', \Gamma' \vdash A' \leq B'$ in the original derivation, the same judgement is proved by transitivity from $c :: \Gamma, \alpha \leq A, \Gamma' \vdash \alpha \leq A'$ and $\Gamma, \alpha \leq A, \Gamma' \vdash A' \leq B'$ in the modified derivation (rule 2, first case). Rules 2 (second case) and 3, 4, and 5 only propagate the replacement inside the derivation.

Lemma 4.6 *Let* $c :: \Gamma, \alpha \leq A, \Gamma' \vdash_+ \alpha \leq A'$ *and* $d :: \Gamma, \alpha \leq A', \Gamma' \vdash_+ B \leq B'$ *be* F-bounded$^+$ *derivations. Then*

$$d[\alpha \leq A' \leftarrow c] :: \Gamma, \alpha \leq A, \Gamma' \vdash_+ B \leq B'.$$

Proof. Routine induction on the structure of d. Cases 1 and 3 entail invoking the Subproof Lemma 4.4. □

The next two lemmata give some properties of the derivations in *F-bounded*$^+$ whose final judgements involve the type *Top*. In particular we show that *Top* is indeed the maximum type with respect to the subtype relation. Furthermore we show that to derive that a type is less than *Top* one must eventually use the rules ($\mathsf{Id} \leq$) or ($\mathsf{Top} \leq$).

Lemma 4.7 *Let $c :: \Gamma \vdash_+ \mathit{Top} \leq A$ be an* F-bounded$^+$ *subtyping derivation. Then $A \equiv \mathit{Top}$ and c must be in the set generated by the following grammar:*

$$e ::= \mathit{Top}_{\Gamma,\mathit{Top}} \mid \mathit{Id}_{\Gamma,\mathit{Top}} \mid e; e.$$

Proof. We proceed by induction on the length $|c|$ of c. Suppose that the thesis holds for $|c| < k$; then, if $|c| = k$, we distinguish various cases according to the last rule in the derivation.

- $(c \equiv \mathit{Id}_{\Gamma,B},\ c \equiv \mathit{Top}_{\Gamma,B})$
  In both cases $A \equiv B \equiv \mathit{Top}$ and thus c is of the desired form.
- $(c \equiv V_{\alpha,B}(c'),\ c \equiv c' \to c'',\ c \equiv \forall\alpha \leq c'.\, c'')$
  Not possible.
- $(c \equiv c'; c'')$
  Since $c' :: \Gamma \vdash_+ \mathit{Top} \leq A'$ and $|c'| < |c|$, by inductive hypothesis, we have that c' must be in the set generated by the grammar and $A' \equiv \mathit{Top}$. Moreover, $c'' :: \Gamma \vdash_+ A' \leq A$, i.e., $c'' :: \Gamma \vdash_+ \mathit{Top} \leq A$. Since $|c''| < |c|$, again by inductive hypothesis, we conclude that $A \equiv \mathit{Top}$ and c'' is generated by the grammar. Therefore $c \equiv c'; c''$ is generated by the grammar. □

Lemma 4.8 *Let $c :: \Gamma \vdash_+ A \leq \mathit{Top}$ be a* F-bounded$^+$ *subtyping derivation. Then c must be in the set generated by the following grammar:*

$$e ::= \mathit{Id}_{\Gamma,\mathit{Top}} \mid \mathit{Top}_{\Gamma,A} \mid d; e,$$

where the variable d ranges over arbitrary derivations.

Proof. We proceed by induction on the length $|c|$ of c. Suppose that the thesis holds for $|c| < k$; then, if $|c| = k$:

- $(c \equiv \mathit{Id}_{\Gamma,B})$
  In this case, by necessity $B \equiv A \equiv \mathit{Top}$ and thus c is of the desired form.
- $(c \equiv \mathit{Top}_{\Gamma,B})$
  In this case, $B \equiv A$ and thus c is of the desired form.
- $(c \equiv V_{\alpha,B}(c'))$
  Not possible, since B should be Top.
- $(c \equiv c' \to c'',\ c \equiv \forall\alpha \leq c'.\, c'')$
  Not possible.
- $(c \equiv c'; c'')$
  In this case $c'' :: \Gamma \vdash_+ A' \leq \mathit{Top}$ and $|c''| < |c|$, hence by inductive hypothesis, c'' is of the desired form and thus also $c \equiv c'; c''$ is. □

The weakening operation, as suggested by its name, allows us to weaken a derivation by adding a new hypothesis to the type or value environments.

Definition 4.9 ***(subtyping weakening)*** Let Γ be $\Gamma_1, \alpha \leq A, \Gamma_2$, with $\Gamma \vdash \Diamond$. The *weakening* of an *F-bounded*$^+$ derivation $c :: \Gamma_1, \Gamma_2 \vdash_+ B \leq C$ with the binding $\alpha \leq A$, denoted by c, Γ (i.e., $c, \Gamma_1, \alpha \leq A, \Gamma_2$) is defined by induction on the structure of c, as follows:[7]

1. $Id_{(\Gamma_1,\Gamma_2),B}, \Gamma = Id_{\Gamma,B}$
2. $V_{\beta,C}(c'), \Gamma = V_{\beta,C}(c', \Gamma)$
3. $Top_{(\Gamma_1,\Gamma_2),B}, \Gamma = Top_{\Gamma,B}$
4. $(c' \to c''), \Gamma = (c', \Gamma) \to (c'', \Gamma)$
5. $(\forall \beta \leq c'.\, c''), \Gamma = \forall \beta \leq (c', (\Gamma, \beta \leq C')).\, (c'', (\Gamma, \beta \leq C'))$
   where $c :: \Gamma_1, \Gamma_2 \vdash_+ (\forall \beta \leq B'.\, B'') \leq (\forall \beta \leq C'.\, C'')$
6. $(c'; c''), \Gamma = (c', \Gamma); (c'', \Gamma)$

Lemma 4.10 *If* $c :: \Gamma_1, \Gamma_2 \vdash_+ B \leq C$ *and* $\Gamma \vdash \Diamond$*, where* $\Gamma \equiv \Gamma_1, \alpha \leq A, \Gamma_2$*, then*

$$c, \Gamma :: \Gamma \vdash_+ B \leq C.$$

Proof. Routine induction on the structure of c. In the $\forall \beta \leq c'.\, c''$ case we also have to prove that $\Gamma, \beta \leq C' \vdash \Diamond$, but this immediately follows from the fact that, by hypothesis, $\Gamma \vdash \Diamond$ and, by Subproof Lemma 4.4, $\Gamma_1, \Gamma_2, \beta \leq C' \vdash \Diamond$. □

Lemma 4.11 (typing weakening) *Let* $\Gamma, \Delta \vdash_+ a : A$ *be an* F-bounded$^+$ *typing judgement. Assuming* $\Gamma, \alpha \leq A' \vdash \Diamond$ *then* $\Gamma, \alpha \leq A', \Delta \vdash_+ a : A$*. Similarly, if* $\Gamma, \Delta, y : B, \Delta' \vdash \Diamond$*, then* $\Gamma, \Delta, y : B, \Delta' \vdash_+ a : A$*.*

Proof. Routine induction. □

4.2.3 Normalization of *F-bounded*$^+$ derivations

In this section, following the ideas proposed for $F_{\leq}$ in [CG92, Pie97], we prove that every *F-bounded*$^+$ subtyping derivation can be transformed into a normal form derivation, where the transitivity rule is not used and the identity rule is used only on variable types. Since every normal form derivation in *F-bounded*$^+$ is also a *dF-bounded* derivation, this result implies that *dF-bounded* is equivalent to *F-bounded*$^+$ and hence to *F-bounded*.

The normalization procedure is presented as a collection of rewrite rules on (linear representations of) subtyping derivations. These rules are separated into three groups. Informally, the rules in the first group push instances of ($\mathsf{Id} \leq$) rule towards the leaves until they are applied to variables or disappear into instances of the ($\mathsf{Top} \leq$) rule. The rules in the second group remove instances of ($\mathsf{Trans} \leq$) that involve identity derivations, and push instances of ($\mathsf{Trans} \leq$) rule towards the leaves until they disappear into instances of the ($\mathsf{TVar} \leq$) rule. The unique rule in the last group removes instances of ($\mathsf{Trans} \leq$) rule that involve Top_A derivations.

[7] As usual, in the De Bruijn notation, when Γ_1, Γ_2 becomes $\Gamma_1, \alpha \leq A, \Gamma_2$, all variable indexes in Γ_2 have to be updated so that they point to the same binder as before. Namely, the index of every free variable in Γ_2 has to be incremented by one.

Definition 4.12 ***(derivation simplification rules)*** The *one step, outermost simplification relation* on subtyping derivations, denoted by $\longrightarrow_o$, is defined by the following rewrite rules.

I. Reflexivity simplification

(1) $Id_{\Gamma, A\to B} \;\longrightarrow_o\; Id_{\Gamma,A} \to Id_{\Gamma,B}$

(2) $Id_{\Gamma,\forall\alpha\leq A.\,B} \;\longrightarrow_o\; \forall\alpha\leq(V_{\alpha,A}(Id_{(\Gamma,\alpha\leq A),A})).\, Id_{(\Gamma,\alpha\leq A),B}$

(3) $Id_{\Gamma,\,Top} \;\longrightarrow_o\; Top_{\Gamma,\,Top}$

II. Cut simplification

(1) $Id_{\Gamma,\alpha}; c \;\longrightarrow_o\; c$

(2) $c; Id_{\Gamma,\alpha} \;\longrightarrow_o\; c$

(3) $V_{\alpha,A}(c); d \;\longrightarrow_o\; V_{\alpha,B}(c; d) \qquad \text{if } d :: \Gamma \vdash A \leq B,\ B \not\equiv Top, \alpha$

(4) $(c_1 \to d_1); (c_2 \to d_2) \;\longrightarrow_o\; c_2; c_1 \to d_1; d_2$

(5) $(\forall\alpha\leq c_1.\, d_1); (\forall\alpha\leq c_2.\, d_2) \;\longrightarrow_o\; \forall\alpha\leq(c_1[\alpha\leq A'\leftarrow c_2]).\,(d_1[\alpha\leq A'\leftarrow c_2]; d_2)$
if $\forall\alpha\leq c_1.\, d_1 :: (\forall\alpha\leq A.\, B) \leq (\forall\alpha\leq A'.\, B')$ and
$\forall\alpha\leq c_2.\, d_2 :: (\forall\alpha\leq A'.\, B') \leq (\forall\alpha\leq A''.\, B'')$

III. Top Cut simplification

(1) $c; Top_{\Gamma,B} \;\longrightarrow_o\; Top_{\Gamma,A} \qquad \text{if } c :: \Gamma \vdash A \leq B$

Hereafter $\longrightarrow$ denotes the "context closure" of the relation $\longrightarrow_o$, which can be defined as the least relation containing $\longrightarrow_o$ and such that, for all derivations c, c', d, if $c \longrightarrow c'$ then:

$(c; d) \longrightarrow (c'; d) \quad$ and $\quad (d; c) \longrightarrow (d; c')$

$V_{\alpha,A}(c) \longrightarrow V_{\alpha,A}(c')$

$(c \to d) \longrightarrow (c' \to d) \quad$ and $\quad (d \to c) \longrightarrow (d \to c')$

$(\forall\alpha\leq c.\, d) \longrightarrow (\forall\alpha\leq c'.\, d) \quad$ and $\quad (\forall\alpha\leq d.\, c) \longrightarrow (\forall\alpha\leq d.\, c')$

The symbol $\longrightarrow^*$ denotes the reflexive and transitive closure of $\longrightarrow$.

Now, to reach the desired result we have to prove three things:

1. every reduction step transforms a derivation of a judgement into another derivation of the same judgement (*subject reduction*);
2. for every *F-bounded*$^+$ derivation, there exists a finite sequence of reduction steps which transforms it into a normal form derivation (*normalization*);
3. every normal form derivation is a *dF-bounded* derivation.

Subject reduction plus normalization imply that, for every *F-bounded*$^+$ derivation, there exists a normal form derivation which proves the same judgement. The third fact completes the proof of the equivalence between *F-bounded*$^+$ and *dF-bounded*.

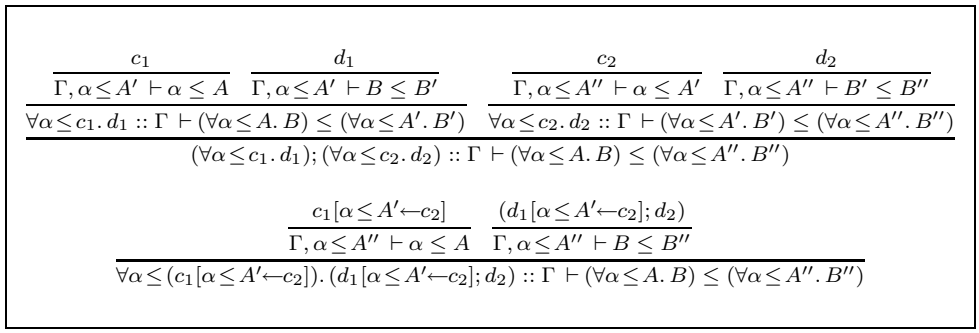

Table 1: A pictorial representation of rule II.5: the two subtyping derivations $(\forall\alpha\leq c_1.\, d_1);(\forall\alpha\leq c_2.\, d_2)$ and $\forall\alpha\leq(c_1[\alpha\leq A'\leftarrow c_2]).\,(d_1[\alpha\leq A'\leftarrow c_2];d_2)$.

Subject reduction for simplification rules

Lemma 4.13 (subject reduction for simplification rules) *If c is a subtyping derivation such that $c :: \Gamma \vdash A \leq B$ and $c \longrightarrow^* d$ then $d :: \Gamma \vdash A \leq B$.*

Proof. First observe that all the simplification rules transform each derivation into a derivation with the same conclusion. The only non trivial case is rule II.5, depicted in Table 1, where Replacement Lemma 4.6 is needed. This gives subject reduction for $\longrightarrow_o$.

Noticing that none of the subtyping rules places any requirement on the shape of the derivations of their hypotheses we can extend the result to $\longrightarrow$ and hence to $\longrightarrow^*$. □

Termination of the normalization procedure

We now prove that every *F-bounded*$^+$ subtyping derivation can be reduced to a normal form in a finite number of steps. As in [CG92, Pie97] the proof relies on the basic observation that when an instance of transitivity is reduced, all new instances of transitivity introduced by the reduction step have a smaller intermediate type.

A derivation of the form $c;d$ is called a *compound derivation*. If $c :: \Gamma \vdash A \leq B$ and $d :: \Gamma \vdash B \leq C$ then B is called the *cut-type* of the derivation, and its (syntactic) length the *cut-size* of the derivation, namely

$$\textit{cut-type}(c;d) = B \qquad\qquad \textit{cut-size}(c;d) = |B|$$

For $X \in \{\mathrm{I}, \mathrm{II}, \mathrm{III}\}$, an *X-redex* in a derivation c is a subderivation of c that can be reduced by using a rule in group X. The result of the reduction is called the *contractum* of the redex. A derivation c is in *X-normal form* if it contains no X-redexes. A derivation in I, II, III-normal form is said to be in *normal form*. By "*innermost II-redex* with cut-size k of a derivation c" we refer to any II-redex d in c such that no proper subderivation of d is a II-redex with cut-size k.

Definition 4.14 ***(rewriting strategy)*** The *rewriting strategy* for the normalization of subtyping derivations comprises the following steps:

1. Perform I-reductions in any order until a I-normal form is reached.

2. If the derivation is not in II-normal form, let k be the largest cut-size of II-redexes, select an innermost redex with cut-size k and reduce it. Then return to step 2.

3. Perform III-reductions in any order until a III-normal form is reached.

First of all, observe that no step in the rewriting strategy generates redexes of the previous steps. Therefore, if we show that each step (separately) terminates then we can conclude that the whole normalization process always terminates, thus producing a derivation in normal form.

Step 1: Notice that rules I.1 and I.2 decrease the size of the type associated with any new I-redex they create, and rule I.3 does not create new I-redexes. Therefore I-rules are strongly normalizing and *Step 1* always terminates.

Step 3: The only rule in *Step 3* strictly decreases the size of the derivation, therefore *Step 3* always terminates as well.

Step 2, outline: The proof of termination for *Step 2* is based on the observation that rules II.1, II.2 do not generate new redexes, while the cut-size of the new redexes generated by rules II.4, II.5 is strictly smaller than the cut-size of the reduced redex. Finally, rule II.3 applied to some redex can generate a new redex with the same cut-size, but it is not difficult to see that any segment of consecutive II.3 reductions can only have a finite length. This is formalized by inserting into the complexity measure of a derivation c a component, called v-complexity, which intuitively represents a bound for the number of possible consecutive II.3 reductions starting from c.

We continue by giving a detailed proof of the termination of Step 2. We first define the *v-complexity* of a derivation c. The *v-complexity* counts, *for every* ";" operator, the number of occurrences of the operator $V_{\cdot,\cdot}(\cdot)$ in its left argument, so that any application of the II.3 rule is guaranteed to decrease this complexity by one. The number of occurrences of V inside c is denoted by $\#_V(c)$.

Definition 4.15 ***(v-complexity)*** The *v-complexity* of a derivation c, denoted by $\#_v(c)$, is defined as follows:

1. $\#_v(Id_{\Gamma,A}) = \#_v(Top_{\Gamma,A}) = 0$;
2. $\#_v(V_{\alpha,A}(c)) = \#_v(c)$;
3. $\#_v((c_1 \to c_2)) = \#_v(c_1) + \#_v(c_2)$;
4. $\#_v((\forall \beta \leq c_1 . c_2)) = \#_v(c_1) + \#_v(c_2)$;
5. $\#_v((c_1; c_2)) = \#_V(c_1) + \#_v(c_1) + \#_v(c_2)$.

Definition 4.16 ***(total complexity)*** The *(total) complexity* of a derivation c, denoted by $comp(c)$, is defined as

$$comp(c) = \langle k, n, \#_v(c) \rangle$$

where k is the maximum cut-size of II-redexes in c, and n is the number of II-redexes with cut-size k in c. Total complexities are ordered lexicographically.

The next two lemmata are useful in proving that each II-reduction decreases the total complexity of a derivation. The first one will be applied to give a characterization of the cut-size of new II-redexes generated by reductions using rule II.5. The second one proves that an application of rule II.3 decreases the v-complexity of a derivation.

Lemma 4.17 *Let* $c :: \Gamma, \alpha \leq A \vdash \alpha \leq A'$ *and* $d :: \Gamma, \alpha \leq A' \vdash B \leq B'$ *be two subtyping derivations. Then the cut-type of any new II-redex in* $d[\alpha \leq A' \leftarrow c]$ *is* A'.

Proof. We proceed by structural induction on d:

- $(d \equiv Id_{(\Gamma,\alpha\leq A'),B'}$ or $d \equiv Top_{(\Gamma,\alpha\leq A'),B'})$
  In this case $d[\alpha \leq A' \leftarrow c]$ does not contain new redexes.

- $(d \equiv V_{\beta,B'}(d_1))$
  We distinguish two subcases. If $\alpha \equiv \beta$ then

  $$d[\alpha \leq A' \leftarrow c] = c; (d_1[\alpha \leq A' \leftarrow c]).$$

  By inductive hypothesis, any new II-redex in $d_1[\alpha \leq A' \leftarrow c]$ has cut-type A'. Moreover the whole derivation $c; (d_1[\alpha \leq A' \leftarrow c])$ can be a new redex and its cut-type is indeed A'.

  If $\alpha \not\equiv \beta$ then

  $$d[\alpha \leq A' \leftarrow c] = V_{\beta,B'}(d_1[\alpha \leq A' \leftarrow c]).$$

  Thus we conclude by inductive hypothesis.

- $(d \equiv d_1 \rightarrow d_2)$
  By definition of replacement

  $$(d_1 \rightarrow d_2)[\alpha \leq A' \leftarrow c] = (d_1[\alpha \leq A' \leftarrow c]) \rightarrow (d_2[\alpha \leq A' \leftarrow c]).$$

  Thus we conclude by inductive hypothesis.

- $(d \equiv \forall \beta \leq d_1.\, d_2)$
  As above.

- $(d \equiv d_1; d_2)$
  By definition of replacement:

  $$(d_1; d_2)[\alpha \leq A' \leftarrow c] = (d_1[\alpha \leq A' \leftarrow c]); (d_2[\alpha \leq A' \leftarrow c]).$$

  First of all notice that if $(d_1[\alpha \leq A' \leftarrow c]); (d_2[\alpha \leq A' \leftarrow c])$ is a II-redex, then also $d_1; d_2$ has to be a II-redex, as can be verified by analyzing Definition 4.5. The length of $(d_1[\alpha \leq A' \leftarrow c]); (d_2[\alpha \leq A' \leftarrow c])$ can be greater than the length of the original redex $d_1; d_2$, but the cut-type remains the same.

  Hence true new redexes can only appear in $d_1[\alpha \leq A' \leftarrow c]$ and $d_2[\alpha \leq A' \leftarrow c]$, but in these cases we conclude by inductive hypothesis. □

Lemma 4.18 *Let l be a subtyping derivation, let $V_{\alpha,A}(c); d$ be a II-redex in l and let l' be the result of replacing $V_{\alpha,A}(c); d$ in l with its contractum $V_{\alpha,B}(c; d)$. Then*

$$\#_v(l') = \#_v(l) - 1.$$

Proof. First observe that, since l' is obtained from l by substituting a subderivation e with a subderivation e' such that $\#_V(e) = \#_V(e')$, then $\#_v(l) - \#_v(l')$ is equal to $\#_v(e) - \#_v(e')$. We can now compute this difference as follows:

$$\begin{aligned}
&\#_v(V_{\alpha,A}(c); d) - \#_v(V_{\alpha,B}(c; d)) \\
&\quad = \#_V(V_{\alpha,A}(c)) + \#_v(V_{\alpha,A}(c)) + \#_v(d) - \#_v(c; d) \\
&\quad = \#_V(c) + 1 + \#_v(c) + \#_v(d) - (\#_V(c) + \#_v(c) + \#_v(d)) \\
&\quad = 1.
\end{aligned}$$

□

We are now ready to prove that rules in group II strictly decrease the total complexity of a derivation and thus that *Step 2* always terminates as well.

Theorem 4.19 *Let l be a subtyping derivation whose maximum cut-size is k. Let $c; d$ be an innermost II-redex in l with cut-size k. Then*

$$comp(l') < comp(l)$$

where l' is the result of replacing the redex $c; d$ in l with its contractum e.

Proof. By cases on the II-rule applied to reduce $c; d$ to e:

- (*Rule II.1*) $c \equiv Id_\alpha$ and $e \equiv d$.
  This reduction removes a II-redex of maximum cut-size from l.

- (*Rule II.2*) $d \equiv Id_\alpha$ and $e \equiv c$.
  Same as *(Rule II.1)*.

- (*Rule II.3*) $c \equiv V_\alpha(c_1)$ and $e \equiv V_\alpha(c_1; d)$.
  This reduction removes a II-redex of maximum cut-size $V_\alpha(c_1); d$ and introduces a new II-redex $c_1; d$ with the same cut-size. However, by Lemma 4.18, $\#_v(l') < \#_v(l)$, hence the total complexity decreases.

- (*Rule II.4*) $c \equiv c_1 \to c_2$, $d \equiv d_1 \to d_2$ and $e \equiv (d_1; c_1) \to (c_2; d_2)$.
  This reduction removes a II-redex of maximum cut-size and may introduce two new redexes $d_1; c_1$ and $c_2; d_2$ with a smaller cut-size.

- (*Rule II.5*) $c \equiv \forall\alpha \le c_1.\, c_2$, $d \equiv \forall\alpha \le d_1.\, d_2$ and $e \equiv \forall\alpha \le c'.\, d'$, where $c' = (c_1[\alpha \le A' \leftarrow d_1])$ and $d' = (c_2[\alpha \le A' \leftarrow d_1]; d_2)$.
  To fix notation, let us suppose that

  $$c :: \Gamma \vdash (\forall\alpha \le A.\, B) \le (\forall\alpha \le A'.\, B')$$

  $$d :: \Gamma \vdash (\forall\alpha \le A'.\, B') \le (\forall\alpha \le A''.\, B'').$$

  The reduction removes a II-redex with maximum cut-size and may add the following new redexes:

  - $(c_2[\alpha \le A' \leftarrow d_1]; d_2)$ with cut-type B';

- new redexes in $c_1[\alpha \leq A' \leftarrow d_1]$ and $c_2[\alpha \leq A' \leftarrow d_1]$, with cut-type A', by Lemma 4.17.

Hence, new redexes have a cut-size smaller than $\mathit{cut\text{-}size}(c; d) = \forall \alpha \leq A'.\, B'$. The old redexes in d_1 are generally copied many times by the substitution operation $c_1[\alpha \leq A' \leftarrow d_1]$, but this replication does not modify the total complexity, since all the redexes in d_1 have a cut-size which is smaller than k (due to the fact that the innermost redex of cut-size k has been chosen); the same considerations apply for $c_2[\alpha \leq A' \leftarrow d_1]$. □

Normal forms are in *dF-bounded*

By the previous results each *F-bounded*$^+$ subtyping derivation can be rewritten, in a finite number of steps, to a normal form derivation that proves the same judgement. We now show that every normal form *F-bounded*$^+$ derivation is a *dF-bounded* derivation, i.e., that it applies reflexivity only to type variables and that it does not use the transitivity rule.

Lemma 4.20 *If $Id_{\Gamma,A} :: \Gamma \vdash A \leq A$ is a normal form subtyping derivation then $A \equiv \alpha$ for some type variable α.*

Proof. Obvious by the form of I-rules. □

Lemma 4.21 *If e is a normal form subtyping derivation then it is not of the shape $c; d$.*

Proof. Let $e \equiv c; d$ be a subtyping derivation and let us show that it is not a normal form.

If c or d are not in I-normal form then obviously e is not in normal form.

Otherwise, if c and d are in I-normal form, we prove the thesis by induction on the length of e. The following table reports a case analysis for c and d, and indicates in each case the reason why $e \equiv c; d$ is not in normal form.

c	d	reason
Id_α	any	rule II.1
any	Id_α	rule II.2
$c_1; c_2$	any	inductive hypothesis
any	$d_1; d_2$	inductive hypothesis
Top_A	$V_{\beta,B}(d_1)$	not possible
Top_A	$d_1 \to d_2$	not possible
Top_A	$\forall \alpha \leq d_1.\, d_2$	not possible
any	Top_A	rule III.1
$V_{\alpha,A}(c_1)$	$V_{\beta,B}(d)$ / $d_1 \to d_2$ / $\forall \alpha \leq d_1.\, d_2$	rule II.3
$c_1 \to c_2$	$V_{\alpha,A}(d_1)$	not possible
$\forall \alpha \leq c_1.\, c_2$	$V_{\beta,B}(d_1)$	not possible
$c_1 \to c_2$	$d_1 \to d_2$	rule II.4
$c_1 \to c_2$	$\forall \alpha \leq d_1.\, d_2$	not possible
$\forall \alpha \leq c_1.\, c_2$	$d_1 \to d_2$	not possible
$\forall \alpha \leq c_1.\, c_2$	$\forall \alpha \leq d_1.\, d_2$	rule II.5

Notice that the only case which is lacking is $V_{\alpha,A}(c_1); d$ with $d :: \Gamma \vdash A \leq Top$ and $d \not\equiv Top_A$ (if $d \equiv Top_A$ the case appears in the table as *any*; Top_A). Now, since d is a I-normal form, Id_{Top} cannot occur in d and thus, by Lemma 4.8, we conclude that $d \equiv c'; c''$, which is not a II-normal form by inductive hypothesis. □

As an immediate corollary we have now the main result of this section.

Theorem 4.22 (transitivity elimination for subtyping) *For every subtyping judgement* $\Gamma \vdash A \leq B$

$$\Gamma \vdash_b A \leq B \quad \textit{iff} \quad \Gamma \vdash_d A \leq B$$

Proof. We already observed that systems *F-bounded*$^+$ and *F-bounded* are equivalent. Moreover we proved that the normalization procedure always transforms an *F-bounded*$^+$ derivation into a normal form derivation of the same judgement, which is, by Lemmata 4.20 and 4.21 a *dF-bounded* derivation. Since *dF-bounded* is a subsystem of *F-bounded*$^+$ this allows us to conclude that also *dF-bounded* and *F-bounded*$^+$ are equivalent, thus proving the thesis. □

Since the subtyping relations in *F-bounded*, *F-bounded*$^+$ and *dF-bounded* coincide, in the following we do not distinguish derivability of subtyping judgement in the three systems unless we need it to refer to the concrete derivation.

5 Type checking

In this section we complete the definition of *dF-bounded*, the algorithmic version of *F-bounded*, by specifying a deterministic set of term formation rules. Then we show that the typing algorithm naturally associated with *dF-bounded* is correct with respect to *F-bounded* and allows us to determine a minimal type for every term which is typable in a given environment.

First of all we introduce the function $\Gamma^{\rightarrow}$, induced by a type environment Γ, which applied to a type A gives back the minimum supertype of A which is an arrow type, when such a supertype exists.

Definition 5.1 Let Γ be a well-formed type environment. Then:

$$\begin{array}{lcll} \Gamma^{\rightarrow}(A \rightarrow B) & = & A \rightarrow B & \textit{if } \Gamma \vdash A \rightarrow B \\ \Gamma^{\rightarrow}(\alpha) & = & \Gamma^{\rightarrow}(\Gamma(\alpha)) & \textit{if } \alpha \in \textit{vars}(\Gamma) \textit{ and } \Gamma(\alpha) \neq \alpha \end{array}$$

Notice that $\Gamma^{\rightarrow}(A)$ is undefined whenever A is *Top*, or a $\forall$ type, or a variable α bounded by itself or by a type B such that $\Gamma^{\rightarrow}(B)$ is undefined. The definition of the minimum $\forall$ supertype of a given type in a type environment is analogous.

Definition 5.2 Let Γ be a well-formed type environment. Then:

$$\begin{array}{lcll} \Gamma^{\forall}(\forall \alpha \leq A.\, B) & = & \forall \alpha \leq A.\, B & \textit{if } \Gamma \vdash \forall \alpha \leq A.\, B \\ \Gamma^{\forall}(\alpha) & = & \Gamma^{\forall}(\Gamma(\alpha)) & \textit{if } \alpha \in \textit{vars}(\Gamma) \textit{ and } \Gamma(\alpha) \neq \alpha \end{array}$$

The reader can easily verify that, if $\Gamma^{\rightarrow}(A)$ is defined, then $\Gamma \vdash A \leq \Gamma^{\rightarrow}(A)$. Similarly, if $\Gamma^{\forall}(A)$ is defined, then $\Gamma \vdash A \leq \Gamma^{\forall}(A)$.

Definition 5.3 The term formation rules of *dF-bounded* are the rules (Var), ($\to$ I), ($\forall$I) of *F-bounded*, plus the rules:

$$\frac{\Gamma, \Delta \vdash f : B \quad \Gamma^{\to}(B) = A' \to B' \quad \Gamma, \Delta \vdash a : A \quad \Gamma \vdash A \leq A'}{\Gamma, \Delta \vdash f(a) : B'} \ (\mathsf{d} \to \mathsf{E})$$

$$\frac{\Gamma, \Delta \vdash f : B \quad \Gamma^{\forall}(B) = \forall \alpha \leq A.\, B' \quad \Gamma \vdash A' \leq A[\alpha \leftarrow A']}{\Gamma, \Delta \vdash f\{A'\} : B'[\alpha \leftarrow A']} \ (\mathsf{d}\forall\mathsf{E})$$

5.1 Correctness

The system *dF-bounded* is correct with respect to *F-bounded*, i.e., all derivable *dF-bounded* judgements are also derivable *F-bounded* judgements.

Theorem 5.4 (correctness) *If* $\Gamma, \Delta \vdash_d a : A$ *then* $\Gamma, \Delta \vdash_b a : A$.

Proof. As proved in the previous section, the subtyping relations in *F-bounded* and *dF-bounded* coincide. Therefore it suffices to observe that for each *dF-bounded* typing rule there exists a typing derivation in *F-bounded* with the same premises and conclusion. Then an inductive reasoning on the structure of the derivation allows us to conclude the proof.

The rules (Var), ($\to$ I), ($\forall$I) are also *F-bounded* rules, thus no consideration is needed. An instance of rule ($\mathsf{d} \to \mathsf{E}$) can be replaced with the *F-bounded* typing derivation:

$$\dfrac{\dfrac{\Gamma, \Delta \vdash f : B \quad \Gamma \vdash B \leq \Gamma^{\to}(B) = A' \to B'}{\Gamma, \Delta \vdash f : A' \to B'} \ (\mathsf{Subs}) \qquad \dfrac{\Gamma, \Delta \vdash a : A \quad \Gamma \vdash A \leq A'}{\Gamma, \Delta \vdash a : A'} \ (\mathsf{Subs})}{\Gamma, \Delta \vdash f(a) : B'} \ (\to\mathsf{E})$$

Finally, the rule ($\mathsf{d}\forall\mathsf{E}$) can be replaced with the *F-bounded* typing derivation:

$$\dfrac{\dfrac{\Gamma, \Delta \vdash f : B \quad \Gamma \vdash B \leq \Gamma^{\forall}(B) = \forall \alpha \leq A.\, B'}{\Gamma, \Delta \vdash f : \forall \alpha \leq A.\, B'} \ (\mathsf{Subs}) \qquad \Gamma \vdash A' \leq A[\alpha \leftarrow A']}{\Gamma, \Delta \vdash f\{A'\} : B'[\alpha \leftarrow A']} \ (\forall\mathsf{E})$$

□

5.2 Completeness and minimal typing

We now prove that the system *dF-bounded* is complete with respect to *F-bounded* in the sense that if there exists a derivation for $\Gamma, \Delta \vdash_b a : A$ in *F-bounded* then we can find a derivation $\Gamma, \Delta \vdash_d a : A'$ in *dF-bounded* such that $\Gamma \vdash A' \leq A$. Moreover, since *dF-bounded* is deterministic, the type A' is uniquely determined and it is a minimum type for the term a in *F-bounded*.

We first need a lemma stating some substitution properties of subtyping derivations which, besides being useful here to prove the completeness result, will be fundamental in the proof of subject reduction for $\beta\eta$ reduction. The lemma informally states that a type variable can be safely replaced with any type satisfying the constraint imposed by the environment.

Let us fix some notation. If $\Gamma \equiv \alpha_1 \leq A_1, \ldots, \alpha_n \leq A_n$ is a type environment, we denote with $\Gamma[\alpha \leftarrow A]$ the type environment obtained from Γ by substituting each

free occurrence of α in the bounds with A, i.e., $\alpha_1 \leq A_1[\alpha \leftarrow A], \ldots, \alpha_n \leq A_n[\alpha \leftarrow A]$. Similarly, if Δ is a value environment we denote by $\Delta[\alpha \leftarrow A]$ the value environment obtained by substituting each free occurrence of α in Δ with the type A.

Lemma 5.5 (type substitution) *Let* $\Gamma, \alpha \leq A, \Gamma' \vdash \diamondsuit$ *and let* $\Gamma, \Gamma'[\alpha \leftarrow A'] \vdash A' \leq A[\alpha \leftarrow A']$.

1. *If* $\Gamma, \alpha \leq A, \Gamma' \vdash B \leq C$ *then*

$$\Gamma, \Gamma'[\alpha \leftarrow A'] \vdash B[\alpha \leftarrow A'] \leq C[\alpha \leftarrow A']$$

2. *If* $\Gamma, \alpha \leq A, \Gamma', \Delta \vdash_b b : B$ *then*

$$\Gamma, \Gamma'[\alpha \leftarrow A'], \Delta[\alpha \leftarrow A'] \vdash_b b[\alpha \leftarrow A'] : B[\alpha \leftarrow A']$$

Proof.

1. The proof is carried out by induction on the structure of the derivation of $\Gamma, \alpha \leq A, \Gamma' \vdash_b B \leq C$ in *F-bounded* and by cases on the last rule used in the derivation. We analyze just the cases of the rules for type variable and bounded quantification.

   - ($\mathsf{Var} \leq$) Let the last rule be:

$$\frac{\Gamma, \alpha \leq A, \Gamma' \vdash \diamondsuit \quad \beta \in \mathit{vars}(\Gamma, \alpha \leq A, \Gamma')}{\Gamma, \alpha \leq A, \Gamma' \vdash \beta \leq (\Gamma, \alpha \leq A, \Gamma')(\beta)} \ (\mathsf{Var} \leq)$$

     We distinguish two subcases:

     - If $(\beta \equiv \alpha)$ then $\beta[\alpha \leftarrow A'] = A'$ and $(\Gamma, \alpha \leq A, \Gamma')(\beta)[\alpha \leftarrow A'] = A[\alpha \leftarrow A']$. Therefore the judgement we want to prove can be written as $\Gamma, \Gamma'[\alpha \leftarrow A'] \vdash A' \leq A[\alpha \leftarrow A']$, which is already present in the hypotheses.
     - If $(\beta \not\equiv \alpha)$ then $(\Gamma, \alpha \leq A, \Gamma')(\beta) = (\Gamma, \Gamma')(\beta)$ and thus the judgement we want to prove becomes
$$\Gamma, \Gamma'[\alpha \leftarrow A'] \vdash \beta \leq (\Gamma, \Gamma')(\beta)[\alpha \leftarrow A'].$$
       Since $\Gamma, \alpha \leq A, \Gamma' \vdash \diamondsuit$, the variable α cannot occur free in Γ, and thus $(\Gamma, \Gamma')(\beta)[\alpha \leftarrow A'] = (\Gamma, \Gamma'[\alpha \leftarrow A'])(\beta)$. Therefore we can construct the following derivation for the desired judgment.
$$\frac{\Gamma, \Gamma'[\alpha \leftarrow A'] \vdash \diamondsuit \quad \beta \in \mathit{vars}(\Gamma, \Gamma'[\alpha \leftarrow A'])}{\Gamma, \Gamma'[\alpha \leftarrow A'] \vdash \beta \leq (\Gamma, \Gamma'[\alpha \leftarrow A'])(\beta)} \ (\mathsf{Var} \leq)$$
       Notice that $\Gamma, \Gamma'[\alpha \leftarrow A'] \vdash \diamondsuit$ follows by Subproof Lemma 4.4, applied to the hypothesis $\Gamma, \Gamma'[\alpha \leftarrow A'] \vdash A' \leq A[\alpha \leftarrow A']$.

   - ($\forall \leq$) Let the last rule be
$$\frac{\Gamma, \alpha \leq A, \Gamma', \beta \leq C' \vdash \beta \leq B' \quad \Gamma, \alpha \leq A, \Gamma', \beta \leq C' \vdash B'' \leq C''}{\Gamma, \alpha \leq A, \Gamma' \vdash \forall \beta \leq B'.\, B'' \leq \forall \beta \leq C'.\, C''} \ (\forall \leq)$$

By Lemma 4.4, we have $\Gamma, \alpha \leq A, \Gamma', \beta \leq C \vdash \Diamond$, and thus it is easy to see that also $\Gamma, \Gamma'[\alpha \leftarrow A'], \beta \leq C'[\alpha \leftarrow A'] \vdash \Diamond$. Hence Lemma 4.10 and the hypothesis $\Gamma, \Gamma'[\alpha \leftarrow A'] \vdash A' \leq A[\alpha \leftarrow A']$ allow us to deduce that $\Gamma, \Gamma'[\alpha \leftarrow A'], \beta \leq C'[\alpha \leftarrow A'] \vdash A' \leq A[\alpha \leftarrow A']$. Therefore, by inductive hypothesis, we have that $\Gamma, \Gamma'[\alpha \leftarrow A'], \beta \leq C'[\alpha \leftarrow A'] \vdash \beta \leq B'[\alpha \leftarrow A']$ and $\Gamma, \Gamma'[\alpha \leftarrow A'], \beta \leq C'[\alpha \leftarrow A'] \vdash B''[\alpha \leftarrow A'] \leq C''[\alpha \leftarrow A']$. Therefore, by using the rule $(\forall \leq)$, we obtain that in the environment $\Gamma, \Gamma'[\alpha \leftarrow A']$

$$(\forall \beta \leq B'[\alpha \leftarrow A']. B''[\alpha \leftarrow A']) \leq (\forall \beta \leq C'[\alpha \leftarrow A']. C''[\alpha \leftarrow A']),$$

that is $\Gamma, \Gamma'[\alpha \leftarrow A'] \vdash (\forall \beta \leq B'. B'')[\alpha \leftarrow A'] \leq (\forall \beta \leq C'. C'')[\alpha \leftarrow A']$.

2. By induction on the structure of the derivation of $\Gamma, \alpha \leq A, \Gamma', \Delta \vdash_b b : B$ in *F-bounded*, and using point (1). □

We are now ready to prove the main theorem of this section.

Theorem 5.6 (completeness and minimal typing) *If* $\Gamma, \Delta \vdash_b a : A$ *then* $\Gamma, \Delta \vdash_d a : A'$ *and* $\Gamma \vdash A' \leq A$.

Proof. By induction on the structure of the derivation of $\Gamma, \Delta \vdash_b a : A$.

We distinguish various cases according to the last rule used in the derivation.

- (Var) Let the last rule be:

$$\frac{\Gamma, \Delta, x : A, \Delta' \vdash_b \Diamond}{\Gamma, \Delta, x : A, \Delta' \vdash_b x : A} \ (\mathsf{Var})$$

  Then this is also a derivation in *dF-bounded*.

- $(\rightarrow \mathsf{I})$ Let the last rule be:

$$\frac{\Gamma, \Delta, x : A \vdash_b b : B}{\Gamma, \Delta \vdash_b \lambda x{:}A. b : A \rightarrow B} \ (\rightarrow \mathsf{I})$$

  By inductive hypothesis there exists a derivation $d :: \Gamma, \Delta, x : A \vdash_d b : B'$, with $\Gamma \vdash B' \leq B$. Then, since $(\rightarrow \mathsf{I})$ is also a *dF-bounded* rule, we obtain the derivation in *dF-bounded*:

$$\frac{d :: \Gamma, \Delta, x : A \vdash_d b : B'}{\Gamma, \Delta \vdash_d \lambda x{:}A. b : A \rightarrow B'} \ (\rightarrow \mathsf{I})$$

  and $\Gamma \vdash A \rightarrow B' \leq A \rightarrow B$ holds by rule $(\rightarrow \leq)$.

- $(\forall \mathsf{I})$ Let the last rule be:

$$\frac{\Gamma, \alpha \leq A, \Delta \vdash_b b : B \quad \alpha \notin FV(\Delta)}{\Gamma, \Delta \vdash_b \Lambda \alpha \leq A. b : \forall \alpha \leq A. B} \ (\forall \mathsf{I})$$

  By inductive hypothesis there exists a derivation $d :: \Gamma, \alpha \leq A, \Delta \vdash_d b : B'$, with $\Gamma, \alpha \leq A \vdash B' \leq B$. Then, since $(\forall \mathsf{I})$ is also a *dF-bounded* rule, we obtain the derivation in *dF-bounded*:

$$\frac{d :: \Gamma, \alpha \leq A, \Delta \vdash_d b : B' \quad \alpha \notin FV(\Delta)}{\Gamma, \Delta \vdash_d (\Lambda\alpha \leq A.\, b) : (\forall \alpha \leq A.\, B')} \ (\forall \mathsf{I})$$

Furthermore, by using rule $(\forall \leq)$, from $\Gamma, \alpha \leq A \vdash B' \leq B$ we can derive $\Gamma \vdash (\forall \alpha \leq A.\, B') \leq (\forall \alpha \leq A.\, B)$.

- (Subs) Let the last rule be:

$$\frac{\Gamma, \Delta \vdash_b a : A \quad \Gamma \vdash A \leq B}{\Gamma, \Delta \vdash_b a : B} \ (\mathsf{Subs})$$

By inductive hypothesis there exists a derivation $d :: \Gamma, \Delta \vdash_d a : A'$, with $\Gamma \vdash A' \leq A$. Thus, by transitivity, $\Gamma \vdash A' \leq B$.

- $(\to \mathsf{E})$ Let the last rule be:

$$\frac{\Gamma, \Delta \vdash_b f : A \to B \quad \Gamma, \Delta \vdash_b a : A}{\Gamma, \Delta \vdash_b f(a) : B} \ (\to \mathsf{E})$$

By inductive hypothesis there exist the derivations

$d_1 :: \Gamma, \Delta \vdash_d f : C$, with $\Gamma \vdash C \leq A \to B$,
$d_2 :: \Gamma, \Delta \vdash_d a : A'$, with $\Gamma \vdash A' \leq A$.

We want to show that $\Gamma \vdash C \leq A \to B$ implies that $\Gamma^{\to}(C)$ is defined and that $\Gamma \vdash \Gamma^{\to}(C) \leq A \to B$. We prove it by induction on the size of the *dF-bounded* derivation of $\Gamma \vdash C \leq A \to B$ and by cases on the last rule used. There are two possibilities. If the last rule is $(\to\leq)$, then C is an arrow type, and the thesis follows immediately from $\Gamma^{\to}(C) = C$. If the rule is $(\mathsf{TVar} \leq)$, then C is a type variable α with $\Gamma \vdash \Gamma(\alpha) \leq A \to B$ and $\Gamma(\alpha) \neq \alpha$. Hence, by induction, $\Gamma^{\to}(\Gamma(\alpha))$, which is equal to $\Gamma^{\to}(\alpha)$, is defined and less than $A \to B$.

Now, let $\Gamma^{\to}(C)$ be $A'' \to B''$. By the shape of *dF-bounded* subtyping rules, $\Gamma \vdash A \leq A''$ and $\Gamma \vdash B'' \leq B$, and thus, by transitivity, $\Gamma \vdash A' \leq A''$. Hence, $\Gamma \vdash f(a) : B''$ can be proved as follows:

$$\frac{d_1 :: \Gamma, \Delta \vdash_d f : C \quad \Gamma^{\to}(C) = A'' \to B'' \quad d_2 :: \Gamma, \Delta \vdash_d a : A' \quad \Gamma \vdash A' \leq A''}{\Gamma, \Delta \vdash_d f(a) : B''} \ (\mathsf{d} \to \mathsf{E})$$

and, as remarked above, $\Gamma \vdash B'' \leq B$.

- $(\forall \mathsf{E})$ Let the last rule be:

$$\frac{\Gamma, \Delta \vdash_b f : \forall \alpha \leq A.\, B \quad \Gamma \vdash_b A' \leq A[\alpha \leftarrow A']}{\Gamma, \Delta \vdash_b f\{A'\} : B[\alpha \leftarrow A']} \ (\forall \mathsf{E})$$

By inductive hypothesis there exists a derivation

$$d :: \Gamma, \Delta \vdash_d f : C, \text{ with } \Gamma \vdash C \leq (\forall \alpha \leq A.\, B).$$

By reasoning as above, we can prove that $\Gamma^{\forall}(C)$ is equal to a type $\forall\alpha\leq A''.\, B''$ such that $\Gamma \vdash (\forall\alpha\leq A''.\, B'') \leq (\forall\alpha\leq A.\, B)$. By the shape of *dF-bounded* subtyping rules, $\Gamma, \alpha\leq A \vdash \alpha \leq A''$ and $\Gamma, \alpha\leq A \vdash B'' \leq B$. By using $\Gamma, \alpha\leq A \vdash \alpha \leq A''$, $\Gamma \vdash A' \leq A[\alpha\leftarrow A']$ (premise of the ($\forall$E) rule) and Lemma 5.5(1), we have $\Gamma \vdash \alpha[\alpha\leftarrow A'] \leq A''[\alpha\leftarrow A']$, i.e.,

$$\Gamma \vdash A' \leq A''[\alpha\leftarrow A'].$$

Therefore, by using the rule (d$\forall$E), we can construct the following *dF-bounded* derivation:

$$\frac{d :: \Gamma, \Delta \vdash_d f : C \quad \Gamma^{\forall}(C) = \forall\alpha\leq A''.\, B'' \quad \Gamma \vdash A' \leq A''[\alpha\leftarrow A']}{\Gamma, \Delta \vdash_d f\{A'\} : B''[\alpha\leftarrow A']}\ (\mathsf{d}\forall\mathsf{E})$$

and, since $\Gamma, \alpha\leq A \vdash B'' \leq B$, recalling that $\Gamma \vdash A' \leq A[\alpha\leftarrow A']$, and by using Lemma 5.5(1) again, we conclude:

$\Gamma \vdash B''[\alpha\leftarrow A'] \leq B[\alpha\leftarrow A']$. □

6 Subject reduction for system *F-bounded*

Subject reduction is one of the primary properties of a typed language. It states that the type is preserved (or sometimes specialized) by the reduction rules of the language and therefore it ensures that a program which has been assigned a type statically, will never go wrong at run-time because of typing errors.

We first need a strengthening lemma stating that unused bindings can be safely discarded from the environment. More precisely, given a judgement $\Gamma(, \Delta) \vdash P$, if a (type or value) variable appearing in the environment does not occur free in P then the corresponding binding in the environment can be removed without affecting the derivability of the judgement.

Lemma 6.1 (strengthening) *1. If the judgement $\Gamma, \alpha\leq A, \Gamma' \vdash B \leq C$ is derivable, $\alpha \notin FV(B) \cup FV(C)$ and $\Gamma, \Gamma' \vdash \diamondsuit$ then also $\Gamma, \Gamma' \vdash B \leq C$ is derivable.*

2. Similarly, if the judgement $\Gamma, \alpha\leq A, \Gamma', \Delta \vdash_b b : B$ is derivable, $\Gamma, \Gamma', \Delta \vdash \diamondsuit$ and $\alpha \notin FV(b) \cup FV(B)$ then also $\Gamma, \Gamma', \Delta \vdash_b b : B$ is derivable.

3. Finally, if $\Gamma, \Delta, x{:}A, \Delta' \vdash b : B$ and $x \notin FV(b)$ then $\Gamma, \Delta, \Delta' \vdash b : B$.

Proof.

1. It is convenient to consider a derivation d for $\Gamma, \alpha\leq A, \Gamma' \vdash B \leq C$ in the deterministic version *dF-bounded* of the system. Then the proof proceeds by straightforward induction on the structure of d and by cases on the last rule used. Only observe that, when treating rule (TVar $\leq$), the well-formedness hypothesis $\Gamma, \Gamma' \vdash \diamondsuit$ ensures that variable α does not occur free in any bound of variables in $vars(\Gamma, \Gamma')$.

2. We first prove a slightly stronger property which only holds for the deterministic variant *dF-bounded* of the system, namely that

$$\Gamma, \alpha \leq A, \Gamma', \Delta \vdash_d b : B \quad \wedge \quad \Gamma, \Gamma', \Delta \vdash \Diamond \quad \wedge \quad \alpha \notin FV(b) \quad \Rightarrow \\ \alpha \notin FV(B) \quad \wedge \quad \Gamma, \Gamma', \Delta \vdash_d b : B.$$

The proof is done by induction on the structure of the *dF-bounded* derivation of the judgement $\Gamma, \alpha \leq A, \Gamma', \Delta \vdash_d b : B$ and by cases on the last rule used.

- (Var) Let the last rule be:

$$\frac{\Gamma, \alpha \leq A, \Gamma', \Delta \vdash_d \Diamond \quad x \in vars(\Delta)}{\Gamma, \alpha \leq A, \Gamma', \Delta \vdash_d x : \Delta(x)} \ (\mathsf{Var})$$

  Since by hypothesis $\Gamma, \Gamma', \Delta \vdash \Diamond$, and $x \in vars(\Delta)$, the judgement $\Gamma, \Gamma', \Delta \vdash_d x : \Delta(x)$ is derivable by using rule (Var). The fact that $\alpha \notin FV(\Delta(x))$ is also an obvious corollary of the well-formedness hypothesis.

- ($\rightarrow$ I) Let the last rule be:

$$\frac{\Gamma, \alpha \leq A, \Gamma', \Delta, x{:}A \vdash_d b : B}{\Gamma, \alpha \leq A, \Gamma', \Delta \vdash_d \lambda x{:}A.\, b : A \rightarrow B} \ (\rightarrow \mathsf{I})$$

  Since $\alpha \notin FV(\lambda x{:}A.\, b)$, clearly

$$(\dagger) \quad \alpha \notin FV(A) \qquad\qquad (\ddagger) \quad \alpha \notin FV(b)$$

  By $\Gamma, \Gamma', \Delta \vdash \Diamond$ and $(\dagger)$, we have that $\Gamma, \Gamma', \Delta, x{:}A \vdash \Diamond$ and thus, by $(\ddagger)$ and inductive hypothesis, we deduce that the variable $\alpha \notin FV(B)$ and $\Gamma, \Gamma', \Delta, x{:}A \vdash_d b : B$ is derivable. Summing up, $\alpha \notin FV(A \rightarrow B)$ and, by using rule ($\rightarrow$ I), the judgement $\Gamma, \Gamma', \Delta \vdash_d \lambda x{:}A.\, b : A \rightarrow B$ is derivable.

- (d $\rightarrow$ E) Let the last rule be:

$$\frac{\Gamma'', \Delta \vdash_d f : B \quad \Gamma''^{\rightarrow}(B) = A' \rightarrow B' \quad \Gamma'', \Delta \vdash_d a : A'' \quad \Gamma'' \vdash A'' \leq A'}{\Gamma'', \Delta \vdash_d f(a) : B'} \ (\mathsf{d} \rightarrow \mathsf{E})$$

  where $\Gamma'' \equiv \Gamma, \alpha \leq A, \Gamma'$. Since $\alpha \notin FV(f(a))$, we have $\alpha \notin FV(f)$ and $\alpha \notin FV(a)$, and therefore, by inductive hypothesis:

$$(a) \quad \alpha \notin FV(B) \qquad\qquad (b) \quad \Gamma, \Gamma', \Delta \vdash_d f : B$$
$$(c) \quad \alpha \notin FV(A'') \qquad\qquad (d) \quad \Gamma, \Gamma', \Delta \vdash_d a : A''$$

  By the fact that $(\Gamma, \alpha \leq A, \Gamma')^{\rightarrow}(B) = A' \rightarrow B'$ and $\Gamma, \Gamma', \Delta \vdash \Diamond$, it is not difficult to see that (a) implies

$$(e) \quad \alpha \notin FV(A' \rightarrow B')$$

  Hence $\alpha \notin FV(A')$ and thus, by $\Gamma, \alpha \leq A, \Gamma' \vdash A'' \leq A'$, (c), $\Gamma, \Gamma' \vdash \Diamond$ and point (1) of this lemma, we have that

$$\Gamma, \Gamma' \vdash A'' \leq A'.$$

  Summing up, the binding $\alpha \leq A$ can be removed from the environment in all the premises of the rule, and thus by using rule (d $\rightarrow$ E) we conclude that $\Gamma, \Gamma', \Delta \vdash_d f(a) : B'$ is derivable. Moreover, by (e), $\alpha \notin FV(B')$.

Rules ($\forall$I) and (d$\forall$E) are treated analogously to ($\to$ I) and (d $\to$ E), respectively. This concludes the proof of the intermediate result.

Now, suppose $\Gamma, \alpha \leq A, \Gamma', \Delta \vdash_b b : B$ derivable in *F-bounded*, $\Gamma, \Gamma', \Delta \vdash \diamondsuit$ and $\alpha \notin FV(b) \cup FV(B)$. By Theorem 5.6 there exists a derivation in *dF-bounded* for $\Gamma, \alpha \leq A, \Gamma', \Delta \vdash_d b : B'$, such that $\Gamma, \alpha \leq A, \Gamma' \vdash B' \leq B$. Hence, by the property of *dF-bounded* just proved

$$(\dagger) \quad \alpha \notin FV(B') \qquad\qquad (\ddagger) \quad \Gamma, \Gamma', \Delta \vdash_d b : B'$$

Since by hypothesis $\alpha \notin FV(B)$, by (†) and point (1) of this lemma, we have $\Gamma, \Gamma' \vdash B' \leq B$. Therefore, by (‡) and using subsumption, we conclude that

$$\Gamma, \Gamma', \Delta \vdash_b b : B.$$

3. Trivial induction on the structure of the derivation. □

It is worth remarking that the absence of a transitivity rule in *dF-bounded* plays a fundamental role, making the proof of point (1) extremely simple. Similarly, the proof of point (2) relies on the possibility of deriving a minimal type for a term in *dF-bounded*, without resorting to subsumption. In fact, notice that the property proved for *dF-bounded* in the proof of point (2) does not hold for the full system. For instance, $\alpha \leq Top, \beta \leq \alpha, x : \alpha \to \alpha \vdash_b x : \beta \to \alpha$, and, although the variable β does not occur free in x, it appears in its type $\beta \to \alpha$.

A basic role in the proof of subject reduction is played by the substitution lemma for types (Lemma 5.5). Furthermore an analogous substitution result for values is needed, stating that a value variable can be safely replaced by any term with the appropriate type.

Lemma 6.2 (value substitution) *Let* $\Gamma, \Delta, x : A, \Delta' \vdash_b b : B$ *and let* $\Gamma, \Delta, \Delta' \vdash_b a : A$. *Then*

$$\Gamma, \Delta, \Delta' \vdash_b b[x \leftarrow a] : B.$$

Proof. The proof proceeds by induction on the structure of the derivation of $\Gamma, \Delta, x : A, \Delta' \vdash_b b : B$ and by cases on the last rule applied.

- (Var) Let the last rule be:

$$\frac{\Gamma, \Delta, x : A, \Delta' \vdash_b \diamondsuit \quad y \in vars(\Delta, x : A, \Delta')}{\Gamma, \Delta, x : A, \Delta' \vdash_b y : B} \ (\mathsf{Var})$$

  If $y \equiv x$ then by necessity $A \equiv B$ and therefore, since $y[x \leftarrow a] = a$, the desired conclusion $\Gamma, \Delta, \Delta' \vdash_b a : A$ is already in the hypotheses. If, on the other hand, $y \not\equiv x$ then $(\Delta, \Delta')(y) = B$. Observing that $\Gamma, \Delta, \Delta' \vdash \diamondsuit$, we conclude $\Gamma, \Delta, \Delta' \vdash_b y : B$ which is exactly the desired conclusion since $y[x \leftarrow a] = y$.

- ($\to$ I) Let the last rule be:

$$\frac{\Gamma, \Delta, x : A, \Delta', y : A' \vdash_b b' : B'}{\Gamma, \Delta, x : A, \Delta' \vdash_b \lambda y{:}A'.\, b' : A' \to B'} \ (\to \mathsf{I})$$

Since we work with De Bruijn terms we can assume without loss of generality that $x \not\equiv y$ and thus that

$$(\lambda y{:}A'.\, b')[x{\leftarrow}a] = \lambda y{:}A'.\, b'[x{\leftarrow}a] \qquad (\dagger)$$

By the Subproof Lemma and Lemma 4.11, $\Gamma, \Delta, \Delta', y{:}A' \vdash_b a : A$. Hence, by inductive hypothesis $\Gamma, \Delta, \Delta', y{:}A' \vdash_b b'[x{\leftarrow}a] : B'$ and therefore, by rule $(\to \mathsf{I})$, we conclude that $\Gamma, \Delta, \Delta' \vdash_b \lambda y{:}A'.\, b'[x{\leftarrow}a] : A' \to B'$. But, recalling $(\dagger)$, this is exactly what we wanted to prove.

- $(\to \mathsf{E})$ Let the last rule be:

$$\frac{\Gamma, \Delta, x{:}A, \Delta' \vdash_b f : A' \to B \quad \Gamma, \Delta, x{:}A, \Delta \vdash_b a' : A'}{\Gamma, \Delta, x{:}A, \Delta' \vdash_b f(a') : B}\ (\to \mathsf{E})$$

By inductive hypothesis the judgements $\Gamma, \Delta, \Delta' \vdash_b f[x{\leftarrow}a] : A' \to B$ and $\Gamma, \Delta, \Delta' \vdash_b a'[x{\leftarrow}a] : A'$ are derivable. Therefore the desired conclusion $\Gamma, \Delta, \Delta' \vdash_b f[x{\leftarrow}a](a'[x{\leftarrow}a]) : B$ follows by rule $(\to \mathsf{E})$.

- (Subs), $(\forall \mathsf{I})$ and $(\forall \mathsf{E})$ are treated as the previous case, by a direct use of the inductive hypothesis. In the case $(\forall \mathsf{I})$, the first statement of Lemma 4.11 must be used. □

Subject reduction is an immediate consequence of the following lemmata, which, in turn, exploits the substitution lemmata for types and values (Lemma 5.5 and Lemma 6.2) and the completeness of the deterministic version of *F-bounded* (Theorem 5.6).

Lemma 6.3 *Let Γ be a type environment and let Δ be a value environment. Then*

1. $\Gamma, \Delta \vdash_b (\lambda x{:}A.\, b)(a) : B \quad \Rightarrow \quad \Gamma, \Delta \vdash_b b[x{\leftarrow}a] : B$;
2. $\Gamma, \Delta \vdash_b (\Lambda \alpha \leq A.\, b)\{A'\} : B \quad \Rightarrow \quad \Gamma, \Delta \vdash_b b[\alpha{\leftarrow}A'] : B$;
3. $\Gamma, \Delta \vdash_b \lambda x{:}A.\, b(x) : B$ *and* $x \notin FV(b) \quad \Rightarrow \quad \Gamma, \Delta \vdash_b b : B$;
4. $\Gamma, \Delta \vdash_b \Lambda \alpha \leq A.\, b\{\alpha\} : B$ *and* $\alpha \notin FV(b) \quad \Rightarrow \quad \Gamma, \Delta \vdash_b b : B$;

Proof.

1. By Theorem 5.6 there exists a derivation $d :: \Gamma, \Delta \vdash_d (\lambda x{:}A.\, b)(a) : B'$ in *dF-bounded* such that $\Gamma \vdash B' \leq B$. This derivation must have the following shape:

$$\frac{\dfrac{\Gamma, \Delta, x{:}A \vdash_d b : B'}{\Gamma, \Delta \vdash_d \lambda x{:}A.\, b : A \to B'}\ (\to \mathsf{I}) \qquad \Gamma, \Delta \vdash_d a : A' \quad \Gamma \vdash A' \leq A}{\Gamma, \Delta \vdash_d (\lambda x{:}A.\, b)(a) : B'}\ (\mathsf{d} \to \mathsf{E})$$

By using subsumption, from $\Gamma, \Delta, x{:}A \vdash_d b : B'$ and $\Gamma \vdash B' \leq B$ we have $\Gamma, \Delta, x{:}A \vdash_b b : B$, and, similarly, from $\Gamma, \Delta \vdash_d a : A'$ and $\Gamma \vdash A' \leq A$ we deduce $\Gamma, \Delta \vdash_b a : A$. Therefore, by Lemma 6.2, we conclude $\Gamma, \Delta \vdash_b b[x{\leftarrow}a] : B$.

2. By Theorem 5.6 there exists a derivation $d :: \Gamma, \Delta \vdash_d (\Lambda\alpha \leq A.\, b)\{A'\} : B'$ in *dF-bounded* such that $\Gamma \vdash B' \leq B$. This derivation must have the following shape:

$$\dfrac{\dfrac{\Gamma, \alpha \leq A, \Delta \vdash_d b : B'' \quad \alpha \notin FV(\Delta)}{\Gamma, \Delta \vdash_d (\Lambda\alpha \leq A.\, b) : (\forall\alpha \leq A.\, B'')}\ (\forall\mathsf{I}) \qquad \Gamma \vdash A' \leq A[\alpha{\leftarrow}A']}{\Gamma, \Delta \vdash_d (\Lambda\alpha \leq A.\, b)\{A'\} : B''[\alpha{\leftarrow}A']}\ (\mathsf{d}\forall\mathsf{E})$$

with $B' \equiv B''[\alpha{\leftarrow}A']$.

By $\Gamma, \alpha \leq A, \Delta \vdash_d b : B''$ and $\Gamma \vdash A' \leq A[\alpha{\leftarrow}A']$ which appear in the derivation, and using Lemma 5.5(2), we have that

$$\Gamma, \Delta \vdash_b b[\alpha{\leftarrow}A'] : B''[\alpha{\leftarrow}A'] \equiv B'.$$

Hence, by using subsumption we conclude $\Gamma, \Delta \vdash_b b[\alpha{\leftarrow}A'] : B$.

3. By Theorem 5.6 there exists a derivation $d :: \Gamma, \Delta \vdash_d \lambda x{:}A.\, b(x) : B'$ in *dF-bounded* such that $\Gamma \vdash B' \leq B$. The derivation d must have the following shape:

$$\dfrac{\dfrac{\Gamma, \Delta, x{:}A \vdash_d b : D \quad \Gamma^{\rightarrow}(D) = A' \rightarrow C \quad \Gamma, \Delta, x{:}A \vdash_d x : A \quad \Gamma \vdash A \leq A'}{\Gamma, \Delta, x{:}A \vdash_d b(x) : C}\ (\mathsf{d}{\rightarrow}\mathsf{E})}{\Gamma, \Delta \vdash_d \lambda x{:}A.\, b(x) : A \rightarrow C}\ (\rightarrow\mathsf{I})$$

where $B' \equiv A \rightarrow C$.

By the Subproof Lemma 4.4 and reflexivity rule we have $\Gamma \vdash C \leq C$, and using $\Gamma \vdash A \leq A'$ we deduce $\Gamma \vdash A' \rightarrow C \leq A \rightarrow C$. Recalling that $\Gamma \vdash D \leq \Gamma^{\rightarrow}(D)$ and $\Gamma^{\rightarrow}(D) = A' \rightarrow C$ we conclude, by transitivity and subsumption, $\Gamma, \Delta, x{:}A \vdash_b b : A \rightarrow C \equiv B'$, and, again by subsumption, $\Gamma, \Delta, x{:}A \vdash_b b : B$. Now, since $x \notin FV(b)$, by strengthening (Lemma 6.1(3)), we reach the desired conclusion $\Gamma, \Delta \vdash_b b : B$.

4. By Theorem 5.6 there exists a derivation $d :: \Gamma, \Delta \vdash_d \Lambda\alpha \leq A.\, b\{\alpha\} : B'$ in *dF-bounded* such that $\Gamma \vdash B' \leq B$. The derivation d must have the following shape:

$$\dfrac{\dfrac{\Gamma, \alpha \leq A, \Delta \vdash_d b : D \quad (\Gamma, \alpha \leq A)^{\forall}(D) = \forall\alpha \leq A'.\, C \qquad \Gamma, \alpha \leq A \vdash \alpha \leq A'}{\Gamma, \alpha \leq A, \Delta \vdash_d b\{\alpha\} : C}\ (\mathsf{d}\forall\mathsf{E}) \qquad \alpha \notin FV(\Delta)}{\Gamma, \Delta \vdash_d \Lambda\alpha \leq A.\, b\{\alpha\} : \forall\alpha \leq A.\, C}\ (\forall\mathsf{I})$$

where $B' \equiv \forall\alpha \leq A.\, C$.

By the Subproof Lemma 4.4 and reflexivity rule we have $\Gamma \vdash C \leq C$, and using $\Gamma, \alpha \leq A \vdash_d \alpha \leq A'$ we deduce $\Gamma \vdash (\forall\alpha \leq A'.\, C) \leq (\forall\alpha \leq A.\, C)$. By Lemma 4.10, $\Gamma, \alpha \leq A \vdash (\forall\alpha \leq A'.\, C) \leq (\forall\alpha \leq A.\, C)$. Therefore, as above, by subsumption we conclude $\Gamma, \alpha \leq A, \Delta \vdash_b b : (\forall\alpha \leq A.\, C)$ and thus, by strengthening (Lemma 6.1(2)), since $\alpha \notin FV(b) \cup FV(\forall\alpha \leq A.\, C)$ and $\Gamma, \Delta \vdash \Diamond$, we have $\Gamma, \Delta \vdash_b b : (\forall\alpha \leq A.\, C)$. Recalling that $B' \equiv \forall\alpha \leq A.\, C$ and $\Gamma \vdash B' \leq B$, by using subsumption, we reach the desired conclusion. □

Now, the theorem of subject reduction for *F-bounded* is an immediate corollary of the previous lemma.

Theorem 6.4 (subject reduction) *Let a be a term in* F-bounded. *If $\vdash_b a : A$ and $a \twoheadrightarrow {}^* a'$ then $\vdash_b a' : A$.*

7 Type Equivalence in system *F-bounded*

Two types mutually related by subtyping are equivalent in the sense that each can be substituted by the other one in any good formation, typing, or subtyping judgement. Having just one type in each equivalence class generally makes a type system slightly easier to use and to understand, both for the programmer and for the theoretician. For this reason, antisymmetry of subtyping is regarded as a desirable property.

The subtype relation in *F-bounded* is *not* antisymmetric, namely, in general, $\Gamma \vdash A \leq B$ and $\Gamma \vdash B \leq A$ does not imply that A and B are (syntactically) the same type. In other words subtype equivalence and (syntactical) equality of types do not coincide in *F-bounded*. For instance we have:

$$\dfrac{\dfrac{\alpha \leq Top \vdash \alpha}{\alpha \leq Top \vdash \alpha \leq \alpha}\ (\mathsf{IdVar}\leq) \quad \dfrac{\alpha \leq Top \vdash Top}{\alpha \leq Top \vdash Top \leq Top}\ (\mathsf{Top}\leq)}{\vdash (\forall \alpha \leq \alpha.\ Top) \leq (\forall \alpha \leq Top.\ Top)}\ (\forall \leq)$$

and also the converse inequality holds:

$$\dfrac{\dfrac{\alpha \leq \alpha \vdash \alpha}{\alpha \leq \alpha \vdash \alpha \leq Top}\ (\mathsf{Top}\leq) \quad \dfrac{\alpha \leq \alpha \vdash Top}{\alpha \leq \alpha \vdash Top \leq Top}\ (\mathsf{Top}\leq)}{\vdash (\forall \alpha \leq Top.\ Top) \leq (\forall \alpha \leq \alpha.\ Top)}\ (\forall \leq)$$

The aim of this section is to characterize type equivalence in *F-bounded* and to suggest how an antisymmetric subtype relation can be recovered. We will see that the above example is paradigmatic, in the sense that, as one would expect, two equivalent types are syntactically the same type up to the replacement of bounds of the kind $\alpha \leq \alpha$ with $\alpha \leq Top$ and vice versa. These considerations will lead to the notion of standard type.

Let us start by giving the formal definition of type equivalence. As pointed out above, two types are equivalent if each one is a subtype of the other.

Definition 7.1 ***(type equivalence)*** Two types A and B are *equivalent in* Γ, written $\Gamma \vdash A \sim B$, if $\Gamma \vdash A \leq B$ and $\Gamma \vdash B \leq A$.

The types A and B are called *equivalent*, written $A \sim B$, if there exists a type environment Γ such that $\Gamma \vdash A \sim B$.

The existential quantification over Γ in the above definition may sound strange. Indeed we will prove later that equivalence does not actually depend on the environment considered but only on the structure of the two types. Namely, whenever $A \sim B$ then $\Gamma \vdash A \sim B$ for any environment Γ such that $\Gamma \vdash A$ and $\Gamma \vdash B$ (Corollary 7.7).

A few simple remarks are in order:

Proposition 7.2 *Let Γ be a type environment, A, B types and let α, β be type variables. Then:*

1. *if* $\Gamma \vdash A \leq \alpha$ *then* A *is a type variable;*

2. $\alpha \sim \beta$ *iff* $\alpha \equiv \beta$*;*

3. *if* $\Gamma, \alpha \leq A \vdash \alpha \leq B$ *and* $B \not\equiv \alpha$, *Top then* $\Gamma, \alpha \leq A \vdash A \leq B$.

Proof.

1. Just consider the structure of a possible derivation of the judgement in *dF-bounded*.

2. Suppose that $\alpha \sim \beta$ (in the environment Γ) and $\alpha \not\equiv \beta$. Since $\Gamma \vdash_d \alpha \leq \beta$ there exists $n > 0$ such that $\Gamma^n(\alpha) = \beta$ and thus β occurs before α in the environment Γ. Therefore it cannot be the case that $\Gamma \vdash_d \beta \leq \alpha$, thus contradicting the hypothesis.

   Conversely, if $\alpha \equiv \beta$ we immediately conclude by using the rule ($\mathsf{IdVar} \leq$).

3. Straightforward, by looking at the shape of the rule ($\mathsf{TVar} \leq$). □

Let us introduce the notion of the standard form for a type. The basic idea is that a bound $\alpha \leq \alpha$ is equivalent to a bound $\alpha \leq Top$, since both essentially correspond to an unbounded quantification.

Definition 7.3 The *standard form* for a type A, denoted by $std(A)$, is defined by induction on the structure of A as follows:

$$
\begin{aligned}
&std(Top) = Top;\\
&std(\alpha) = \alpha;\\
&std(A \to B) = std(A) \to std(B);\\
&std(\forall \alpha \leq A.\, B) = \begin{cases} \forall \alpha \leq std(A).\, std(B) & \text{if } A \not\equiv \alpha \\ \forall \alpha \leq Top.\, std(B) & \text{if } A \equiv \alpha \end{cases}
\end{aligned}
$$

We first prove that every type is equivalent to its standard form and that two types with the same standard form are equivalent.

Lemma 7.4 *Let* Γ *be a type environment and let* A *be a type. If* $\Gamma \vdash A$ *then* $\Gamma \vdash A \sim std(A)$.

Proof. We prove by induction on the structure of A that $\Gamma \vdash A \leq std(A)$ and $\Gamma \vdash std(A) \leq A$.

- $A \equiv \alpha$, *Top*
  Immediate, by rule ($\mathsf{Id} \leq$).

- $A \equiv A' \to A''$
  Trivial induction.

- $A \equiv \forall \alpha \leq A'.\, A''$
  The hypothesis $\Gamma \vdash \forall \alpha \leq A'.\, A''$ implies that

$$\Gamma, \alpha \leq A' \vdash A' \qquad \text{and} \qquad \Gamma, \alpha \leq A' \vdash A''$$

and thus, since $FV(std(A)) = FV(A)$, we easily conclude that:

$$\Gamma, \alpha \leq std(A') \vdash A' \qquad \text{and} \qquad \Gamma, \alpha \leq std(A') \vdash A''.$$

By inductive hypothesis we have:

(1) $\Gamma, \alpha \leq A' \vdash A' \sim std(A')$ (3) $\Gamma, \alpha \leq std(A') \vdash A' \sim std(A')$
(2) $\Gamma, \alpha \leq A' \vdash A'' \sim std(A'')$ (4) $\Gamma, \alpha \leq std(A') \vdash A'' \sim std(A'')$

Now, if $A' \not\equiv \alpha, Top$ then, by definition, $std(A) = \forall \alpha \leq std(A').\, std(A'')$ and $std(A') \not\equiv \alpha, Top$. Thus we can construct a derivation for $\Gamma \vdash std(A) \leq A$ as follows:

$$\dfrac{\dfrac{\Gamma, \alpha \leq A' \vdash A' \leq std(A')(1) \quad A' \not\equiv \alpha, Top \quad std(A') \not\equiv \alpha, Top}{\Gamma, \alpha \leq A' \vdash \alpha \leq std(A')}\ (\mathsf{TVar}\leq) \qquad \Gamma, \alpha \leq A' \vdash std(A'') \leq A''(2)}{\Gamma \vdash (\forall \alpha \leq std(A').\, std(A'')) \leq (\forall \alpha \leq A'.\, A'')}\ (\forall \leq)$$

and similarly for $\Gamma \vdash A \leq std(A)$:

$$\dfrac{\dfrac{\Gamma, \alpha \leq std(A') \vdash std(A') \leq A'(3) \quad A', std(A') \not\equiv \alpha, Top}{\Gamma, \alpha \leq std(A') \vdash \alpha \leq A'}\ (\mathsf{TVar}\leq) \qquad \Gamma, \alpha \leq std(A') \vdash A'' \leq std(A'')(4)}{\Gamma \vdash (\forall \alpha \leq A'.\, A'') \leq (\forall \alpha \leq std(A').\, std(A''))}\ (\forall \leq)$$

If $A' \equiv Top$ or $A' \equiv \alpha$ we cannot use rule ($\mathsf{TVar} \leq$), and in both cases $std(A) = \forall \alpha \leq Top.\, std(A'')$. The case $A' \equiv Top$, can be treated by substituting the instances of rule ($\mathsf{TVar} \leq$) in both derivations by instances of rule ($\mathsf{Top} \leq$). The case $A' \equiv \alpha$ is managed by replacing the instances of rule ($\mathsf{TVar} \leq$) in the two derivations, by an instance of rule ($\mathsf{Top} \leq$) in the first one and by an instance of ($\mathsf{Id} \leq$) in the second one. □

Corollary 7.5 *If $\Gamma \vdash A$, $\Gamma \vdash B$, and $std(A) = std(B)$ then $\Gamma \vdash A \sim B$ and thus $A \sim B$.*

Proof. By the previous lemma, $\Gamma \vdash A \sim std(A)$ and $\Gamma \vdash std(B) \sim B$. By $std(A) = std(B)$ and by transitivity, $\Gamma \vdash A \sim B$; hence $A \sim B$. □

We can now prove the inverse implication, namely the fact that equivalent *F-bounded* types have the same normal form. Here we make a crucial use of the result of transitivity elimination. This explains why such a property, which is by now easy, was claimed but not proved in previous works [Kat92, Ghe97].

Proposition 7.6 *Let A and B be types. If $A \sim B$ then $std(A) = std(B)$.*

Proof. Let Γ be an environment such that $\Gamma \vdash A \sim B$, that is:

$$\Gamma \vdash_d A \leq B \text{ and } \Gamma \vdash_d B \leq A, \qquad (\dagger)$$

where the subtyping derivations are assumed to be in *dF-bounded*. We proceed by induction on the structure of A.

- $(A \equiv \alpha)$
  In this case, by Proposition 7.2, we conclude $B \equiv \alpha$.

- $(A \equiv \mathit{Top})$
  In this case, by Lemma 4.7, we conclude that $B \equiv \mathit{Top}$.

- $(A \equiv A' \to A'')$
  In this case $B \equiv B' \to B''$, since the last rules used in the derivations for (†) are necessarily instances of $(\to\leq)$. Moreover, from the shape of this, we immediately get that $\Gamma \vdash A' \sim B'$ and $\Gamma \vdash A'' \sim B''$. Therefore we conclude by inductive hypothesis.

- $(A \equiv \forall\alpha \leq A'.\, A'')$
  Reasoning as above, the only rule that allows one to prove (†) is $(\forall \leq)$ and thus we have $B \equiv \forall\alpha \leq B'.\, B''$ and

  $$\begin{array}{llll} (a) & \Gamma, \alpha \leq B' \vdash \alpha \leq A' & (c) & \Gamma, \alpha \leq A' \vdash \alpha \leq B' \\ (b) & \Gamma, \alpha \leq B' \vdash A'' \leq B'' & (d) & \Gamma, \alpha \leq A' \vdash B'' \leq A'' \end{array}$$

  By Lemma 4.6, and (c), (b) we conclude that

  $$(e)\ \Gamma, \alpha \leq A' \vdash A'' \leq B''.$$

  From (d) and (e) we have that $A'' \sim B''$ and therefore, by inductive hypothesis, $\mathit{std}(A'') = \mathit{std}(B'')$.

  Now, if A' and B' are both different from α and *Top* then, by Proposition 7.2, point (3) and (a), (c) we have

  $$\text{(f)}\ \Gamma, \alpha \leq B' \vdash B' \leq A' \quad \text{and} \quad \text{(g)}\ \Gamma, \alpha \leq A' \vdash A' \leq B'.$$

  By (g), Lemma 4.6 and (a) we deduce

  $$(h)\ \Gamma, \alpha \leq B' \vdash A' \leq B',$$

  and thus $A' \sim B'$. Therefore, by inductive hypothesis $\mathit{std}(A') = \mathit{std}(B')$, and we conclude $\mathit{std}(A) = \mathit{std}(B)$.

  If, on the other hand, $A' \equiv \alpha$, *Top* then the only rules that allow us to obtain the conclusion (c) are $(\mathsf{IdVar} \leq)$ or $(\mathsf{Top} \leq)$ and thus $B' \equiv \alpha$ or $B' \equiv \mathit{Top}$. Summing up, and, reasoning by symmetry, we have that $A' \equiv \alpha$ or $A' \equiv \mathit{Top}$ if and only if $B' \equiv \alpha$ or $B' \equiv \mathit{Top}$. Therefore, in this case too we conclude $\mathit{std}(A) = (\forall\alpha \leq \mathit{Top}.\, \mathit{std}(A'')) = (\forall\alpha \leq \mathit{Top}.\, \mathit{std}(B'')) = \mathit{std}(B)$. □

As an immediate corollary of the previous lemma and of Corollary 7.5, we obtain the independence of type equivalence from the environment.

Corollary 7.7 *If $A \sim B$, $\Gamma \vdash A$ and $\Gamma \vdash B$, then $\Gamma \vdash A \sim B$.*

The result of this section suggests a very simple way to obtain a formulation of *F-bounded* in which the subtyping relation is antisymmetric, which consists of forbidding types with shape $\forall\alpha \leq \alpha.\, A$, where a type variable has the variable itself as bound. Such a system contains exactly one representative for each class of equivalent *F-bounded* types.

Proposition 7.8 *Let* F-bounded strict *be the system having the same rules as those of* F-bounded*, but with the constraint that the bound of a variable must be different from the variable itself. Then the following facts hold.*

1. Conservativity*: F-bounded is a conservative extension of* F-bounded strict*.*
2. Fullness*: for any type A in* F-bounded *there exists A' in* F-bounded strict *such that $A \sim A'$ (in system* F-bounded*).*
3. Antisymmetry*: for any A, A' in* F-bounded strict*, if $A \sim A'$ then $A \equiv A'$.*

Proof. As for (1), just observe that any algorithmic derivation in *F-bounded* which does not contain any $\alpha \leq \alpha$ bound in the conclusion, does not contain any $\alpha \leq \alpha$ bound anywhere else. To prove (2) let A' be $std(A)$. Finally (3) follows by observing that $A = std(A)$, $A' = std(A')$ and that, by Proposition 7.6, $std(A) = std(A')$. □

8 Other formulations of system *F-bounded*

The (sub)typing rules of *F-bounded* closely correspond to $F_{\leq}$ rules, with the only exception being $(\forall \leq)$. In fact, the most immediate generalization of the rule $(\forall \leq)$ of $F_{\leq}$ would be as follows:

$$\frac{\Gamma, \alpha \leq A' \vdash A' \leq A \quad \Gamma, \alpha \leq A' \vdash B \leq B'}{\Gamma \vdash (\forall \alpha \leq A.\, B) \leq (\forall \alpha \leq A'.\, B')} \ (\forall' \leq)$$

In this section we study the variant of *F-bounded* including rule $(\forall' \leq)$ instead of $(\forall \leq)$ and we prove some properties first conjectured in [Kat92] and later in [Ghe97]. More precisely, we first show that rule $(\forall' \leq)$ is strictly less powerful than the $(\forall \leq)$ rule adopted in this paper. Then we prove that the two rules are equivalent if we either forbid $\alpha \leq \alpha$ bounds, or we add also the following rule to the system:

$$\frac{\Gamma, \alpha \leq \alpha \vdash A}{\Gamma \vdash (\forall \alpha \leq \alpha.\, A) \leq (\forall \alpha \leq \mathit{Top}.\, A)} \ (\forall \mathsf{Top} \leq)$$

The next proposition shows that rule $(\forall' \leq)$ above is strictly less expressive than rule $(\forall \leq)$. In fact it does not allow one to prove, for example, the judgement $\vdash (\forall \alpha \leq \alpha.\, \mathit{Top}) \leq (\forall \alpha \leq \mathit{Top}.\, \mathit{Top})$, which is derivable in *F-bounded*.

Proposition 8.1 *In a system obtained from* F-bounded *by substituting rule $(\forall \leq)$ with $(\forall' \leq)$, one cannot prove the judgement $\Gamma \vdash (\forall \alpha \leq A.\, B) \leq (\forall \alpha \leq \mathit{Top}.\, B')$, for any choice of Γ, B, B' and $A \not\equiv \mathit{Top}$.*

Proof. We prove it by reduction ad absurdum. Suppose that derivations of such judgements exist in the system, and let k be the minimum height of such derivations. A derivation of height k cannot end with an instance of $(\mathsf{Trans} \leq)$ rule, otherwise (at least) one of the two premises would have the desired shape and a derivation with a height smaller than k. In fact, it is easy to verify that the intermediate type would have to be a $\forall$ type with shape $\forall \alpha \leq A''.\, B''$, and thus, if $A'' \equiv \mathit{Top}$ then the left subderivation (or otherwise the right one) would have a judgement of the desired shape as its conclusion. Hence the judgement must have been proved by rule $(\forall' \leq)$. This implies that $\Gamma, \alpha \leq \mathit{Top} \vdash \mathit{Top} \leq A$. Now, it is easy

to see that Lemma 4.7 still holds for this variant of *F-bounded* (the proof remains the same since it does not depend on the formulation of rule $(\forall \leq)$) and therefore we conclude that $A \equiv Top$, thus contradicting the hypothesis. □

We now show that equivalence can be regained by either restricting the types or by adding the rule $(\forall \mathsf{Top} \leq)$ above. We first give a name to the systems corresponding to these different choices.

Definition 8.2 ***(F-bounded$_\leq$ and F-bounded$^-$)*** The system *F-bounded*$_\leq$ is obtained from *F-bounded* by replacing the rule $(\forall \leq)$ with $(\forall' \leq)$ and adding the rule $(\forall \mathsf{Top} \leq)$.

The system *F-bounded*$^-$ is obtained from *F-bounded* by replacing the rule $(\forall \leq)$ with $(\forall' \leq)$ and forbidding $\alpha \leq \alpha$ bounds, where a variable is a bound for itself.

Notation 8.3 When necessary to avoid ambiguity, a judgement derivable in *F-bounded*$_\leq$ and *F-bounded*$^-$ will be respectively denoted as

$$Pre \vdash_{b\leq} Concl \qquad \text{and} \qquad Pre \vdash_{-} Concl$$

Proposition 8.4 *The systems* F-bounded *and* F-bounded$_\leq$ *are equivalent, i.e.,*

$$\Gamma, \Delta \vdash_b a : A \quad \textit{iff} \quad \Gamma, \Delta \vdash_{b\leq} a : A$$

Proof. Since the two systems have the same term formation rules, it is sufficient to show that for all subtyping judgements:

$$\Gamma \vdash_b A \leq B \quad \text{iff} \quad \Gamma \vdash_{b\leq} A \leq B$$

($\Rightarrow$) Let us consider a derivation $d :: \Gamma \vdash_d A \leq B$ in the deterministic system *dF-bounded*.[8] We show by induction on d that it can be transformed into a derivation $d_\leq :: \Gamma \vdash_{b\leq} A \leq B$. We distinguish various cases according to the last rule applied in the derivation d.

- $(\mathsf{IdVar} \leq)$, $(\rightarrow\leq)$, $(\mathsf{Top} \leq)$: Just notice that such rules are (instances of) *F-bounded*$_\leq$ rules, and apply the inductive hypothesis to the premises.
- $(\mathsf{TVar} \leq)$ The derivation is of the kind

$$\dfrac{\dfrac{d'}{\Gamma \vdash_d \Gamma(\alpha) \leq B} \quad B \not\equiv \alpha, Top \quad \Gamma(\alpha) \neq \alpha, Top}{\Gamma \vdash_d \alpha \leq B}\ (\mathsf{TVar}\leq)$$

  By inductive hypothesis we can obtain the *F-bounded*$_\leq$ derivation $d'_\leq$ and thus:

$$\dfrac{\dfrac{\Gamma \vdash \Diamond}{\Gamma \vdash_{b\leq} \alpha \leq \Gamma(\alpha)}\ (\mathsf{Var}\leq) \quad \dfrac{d'_\leq}{\Gamma \vdash_{b\leq} \Gamma(\alpha) \leq B}}{\Gamma \vdash_{b\leq} \alpha \leq B}\ (\mathsf{Trans}\leq)$$

- $(\forall \leq)$ The derivation is of the kind

[8] Basically the same proof can be carried out within the non deterministic version.

$$\frac{\dfrac{d_1}{\Gamma, \alpha \leq A' \vdash_d \alpha \leq A} \quad \dfrac{d_2}{\Gamma, \alpha \leq A' \vdash_d B \leq B'}}{\Gamma \vdash_d (\forall \alpha \leq A.\, B) \leq (\forall \alpha \leq A'.\, B')}\ (\forall \leq)$$

We distinguish three cases, according to the shape of the type A and in each case we give the *F-bounded*$_{\leq}$ derivation for $\Gamma \vdash_{b\leq} (\forall \alpha \leq A.\, B) \leq (\forall \alpha \leq A'.\, B')$

- $(A \equiv Top)$

$$\frac{\dfrac{\Gamma, \alpha \leq A' \vdash_{b\leq} A'}{\Gamma, \alpha \leq A' \vdash_{b\leq} A' \leq Top}\ (\mathsf{Top}\leq) \quad \dfrac{d_{2\leq}}{\Gamma, \alpha \leq A' \vdash_{b\leq} B \leq B'}\ (\text{ind.hyp.})}{\Gamma \vdash_{b\leq} (\forall \alpha \leq Top.\, B) \leq (\forall \alpha \leq A'.\, B')}\ (\forall' \leq)$$

- $(A \equiv \alpha)$

$$\frac{\dfrac{\Gamma, \alpha \leq \alpha \vdash_{b\leq} B}{\Gamma \vdash_{b\leq} (\forall \alpha \leq \alpha.\, B) \leq (\forall \alpha \leq Top.\, B)}\ (\forall \mathsf{Top}\leq) \quad d}{\Gamma \vdash_{b\leq} (\forall \alpha \leq \alpha.\, B) \leq (\forall \alpha \leq A'.\, B')}\ (\mathsf{Trans}\leq)$$

  where d is the derivation of the previous case.

- $(A \not\equiv \alpha, Top)$
  In this case the derivation d_1 is necessarily of the kind:

$$\frac{\dfrac{d'_1}{\Gamma, \alpha \leq A' \vdash_d A' \leq A}}{\Gamma, \alpha \leq A' \vdash_d \alpha \leq A}\ (\mathsf{TVar}\leq)$$

  Therefore we can construct the *F-bounded*$_{\leq}$ derivation:

$$\frac{\dfrac{d'_{1\leq}}{\Gamma, \alpha \leq A' \vdash_{b\leq} A' \leq A}\ (\mathsf{hyp.ind}) \quad \dfrac{d_{2\leq}}{\Gamma, \alpha \leq A' \vdash_{b\leq} B \leq B'}\ (\text{ind.hyp.})}{\Gamma \vdash_{b\leq} (\forall \alpha \leq A.\, B) \leq (\forall \alpha \leq A'.\, B')}\ (\forall' \leq)$$

($\Leftarrow$) Just notice that all *F-bounded*$_{\leq}$ rules are also *F-bounded* rules, with the exception of $(\forall' \leq)$ and $(\forall \mathsf{Top} \leq)$ rules which can be transformed into *F-bounded* derivations with the same premises and conclusion. For the $(\forall' \leq)$ rule the corresponding derivation is:

$$\frac{\dfrac{\dfrac{\Gamma, \alpha \leq A' \vdash \Diamond}{\Gamma, \alpha \leq A' \vdash_b \alpha \leq A'}\ (\mathsf{Var}\leq) \quad \Gamma, \alpha \leq A' \vdash_b A' \leq A}{\Gamma, \alpha \leq A' \vdash_b \alpha \leq A}\ (\mathsf{Trans}\leq) \quad \Gamma, \alpha \leq A' \vdash_b B \leq B'}{\Gamma \vdash_b (\forall \alpha \leq A.\, B) \leq (\forall \alpha \leq A'.\, B')}\ (\forall \leq)$$

while for the $(\forall \mathsf{Top} \leq)$ rule the corresponding derivation is:

$$\frac{\dfrac{\Gamma, \alpha \leq Top \vdash_b \alpha}{\Gamma, \alpha \leq Top \vdash_b \alpha \leq \alpha}\ (\mathsf{Id}\leq) \quad \dfrac{\Gamma, \alpha \leq Top \vdash_b A}{\Gamma, \alpha \leq Top \vdash_b A \leq A}\ (\mathsf{Id}\leq)}{\Gamma \vdash_b (\forall \alpha \leq \alpha.\, A) \leq (\forall \alpha \leq Top.\, A)}\ (\forall \leq)$$

It would be easy to show, by exploiting the Subproof Lemma 4.4, that all the good formation premises in the two derivations hold. □

Since not all *F-bounded* judgements are acceptable in *F-bounded*$^-$, we cannot prove the equivalence of the two systems, but only that the first one is a conservative extension of the second one.

Proposition 8.5 *The system* F-bounded *is a conservative extension of the system* F-bounded$^-$*, i.e.,*

$$\Gamma, \Delta \vdash_b a : A \quad \textit{iff} \quad \Gamma, \Delta \vdash_- a : A$$

for any judgement $\Gamma, \Delta \vdash a : A$ *not containing* $\alpha \leq \alpha$ *bounds.*

Proof. Since the two systems have the same term formation rules, it suffices to show that:

$$\Gamma \vdash_b A \leq B \quad \text{iff} \quad \Gamma \vdash_- A \leq B$$

for any *F-bounded*$^-$ judgement $\Gamma \vdash A \leq B$.

The proof is very similar to that of Proposition 8.4. In the ($\Rightarrow$) part the only difference is that, for the ($\forall \leq$) rule, the case $A \equiv \alpha$ cannot arise since *F-bounded*$^-$ judgements do not contain $\alpha \leq \alpha$ bounds. Thus the ($\forall \mathsf{Top} \leq$) rule is not needed. As for the ($\Leftarrow$) part, it suffices to remove the treatment of the rule ($\forall \mathsf{Top} \leq$). □

To conclude we remark that the alternative formulations analyzed in this section differ essentially in the treatment of the $\alpha \leq \alpha$ bound. If one believes that the $\alpha \leq \alpha$ bound should be considered different from the $\alpha \leq \mathit{Top}$ bound, then the system of choice should contain the ($\forall' \leq$) rule and no ($\forall \mathit{Top} \leq$) rule. This system is strictly less expressive than *F-bounded*, but its subtype relation is antisymmetric (the proof of this fact is simple, but it does not appear in this paper). If one believes that the $\alpha \leq \alpha$ bound is just an equivalent way of expressing the $\alpha \leq \mathit{Top}$ constraint, the most reasonable choice is to disallow this kind of bound altogether, namely *F-bounded*$^-$ is the right system. In this way, the system obtained is antisymmetric and no type is lost, i.e., for every *F-bounded* type there is (exactly) one equivalent type with no $\alpha \leq \alpha$ bounds. Moreover, with this limitation, the two different formulations of the $\forall$ subtyping rule turn out to be equivalent. Finally, if one is interested in studying the variant where the greatest amount of terms can be written down and typed, the one we called *F-bounded* is the system of choice.

9 Conservativity with respect to $\mathbf{F}_{\leq}$

In this section we show that *F-bounded* is a conservative extension of $F_{\leq}$. As an outcome, two results proved in the literature for $F_{\leq}$, namely undecidability of (sub)typing [Pie94] and non-conservativity of strong recursive types [Ghe93], can be easily extended to *F-bounded*.

We consider here the algorithmic version $\mathit{Alg}F_{\leq}$ of $F_{\leq}$, as defined in [CG92]. As discussed in the introduction, system $F_{\leq}$ differs from *F-bounded* essentially because the first one does not allow a type variable to occur in its own bound. Formally, the operation $FV_{\leq}$ that gives back the free variables of a type is defined as in Subsection 3.2, with the exception of the clause for the $\forall$ types, which becomes $FV_{\leq}(\forall \alpha \leq A.\, B) = FV_{\leq}(A) \cup (FV_{\leq}(B) \setminus \{\alpha\})$.

Referring to Section 3, the definitions of types, terms, environments, and judgements are the same. The rule (TEnv) changes as follows.

$$\frac{\Gamma \vdash \Diamond \quad FV_{\leq}(A) \subseteq \mathit{vars}(\Gamma)}{\Gamma, \alpha \leq A \vdash \Diamond} \ (\mathsf{TEnv}_{\mathsf{sub}})$$

Finally, the subtyping and typing rules are the same as those of *dF-bounded*, the algorithmic version of *F-bounded* (Definitions 4.1 and 5.3), with the exception of the subtyping rule for $\forall$ types and the rule of $\forall$ elimination which change as follows.

$$\frac{\Gamma \vdash A' \leq A \quad \Gamma, \alpha \leq A' \vdash B \leq B'}{\Gamma \vdash (\forall \alpha \leq A.\, B) \leq (\forall \alpha \leq A'.\, B')} \ (\forall \leq_{\mathsf{sub}})$$

$$\frac{\Gamma, \Delta \vdash f : B \quad \Gamma^{\forall}(B) = \forall \alpha \leq A.\, B' \quad \Gamma \vdash A' \leq A}{\Gamma, \Delta \vdash f\{A'\} : B'[\alpha \leftarrow A']} \ (\forall \mathsf{E}_{\mathsf{sub}})$$

Notation 9.1 When necessary to avoid ambiguity, a judgement derivable in *(Alg)*$F_{\leq}$ will be denoted as

$$Pre \vdash_{\leq} Concl.$$

The first basic property to observe about system $F_{\leq}$ is the fact that, if $A' \leq A''$ can be derived in an environment $\Gamma, \alpha \leq A$, and α does not occur free either in A' or in A'', then the result can be strengthened by removing the hypothesis $\alpha \leq A$ from the environment. A similar result has already been proved for system *F-bounded* in Lemma 6.1.

Lemma 9.2 *Let* $\Gamma, \alpha \leq A \vdash_{\leq} A' \leq A''$ *and suppose* $\alpha \notin FV(A') \cup FV(A'')$. *Then* $\Gamma \vdash_{\leq} A' \leq A''$.

Proof. Trivial induction. □

The main difference between the algorithmic versions of *F-bounded* and $F_{\leq}$ resides in the first premise of the $(\forall \leq)$ rule; hence the next lemma is the key of the conservativity proof. It states essentially that the two premises are equivalent if we restrict ourselves to $F_{\leq}$ types.

Lemma 9.3 *If* $\Gamma, \alpha \leq A \vdash_{\leq} \alpha \leq A'$ *and* $\alpha \notin FV(A')$ *then* $\Gamma \vdash_{\leq} A \leq A'$

Proof. Let us consider the last rule applied in the derivation. It can be neither $(\mathsf{IdVar} \leq)$, otherwise $A' \equiv \alpha$, nor $(\rightarrow \leq)$, nor $(\forall \leq_{\mathsf{sub}})$. Therefore only two cases can arise:

- $(\mathsf{TVar} \leq)$ In this case the derivation has the shape:

$$\frac{\dfrac{d'}{\Gamma, \alpha \leq A \vdash_{\leq} A \leq A'}}{\Gamma, \alpha \leq A \vdash_{\leq} \alpha \leq A'} \ (\mathsf{TVar} \leq)$$

  By $F_{\leq}$ notion of well-formedness for type environments, $\alpha \notin FV(A)$ and by hypothesis $\alpha \notin FV(A')$. Hence, by Lemma 9.2, $d' :: \Gamma, \alpha \leq A \vdash_{\leq} A \leq A'$ can be strengthened to $d'' :: \Gamma \vdash_{\leq} A \leq A'$.

- $(\mathsf{Top} \leq)$ In this case $A' \equiv Top$ and the derivation has the shape:

$$\frac{\Gamma, \alpha \leq A \vdash_{\leq} \alpha}{\Gamma, \alpha \leq A \vdash_{\leq} \alpha \leq Top} \ (\mathsf{Top} \leq)$$

By $\Gamma, \alpha \leq A \vdash_{\leq} \diamondsuit$, we have $\Gamma \vdash_{\leq} \diamondsuit$ and $FV(A) \subseteq vars(\Gamma)$. Therefore $\Gamma \vdash_{\leq} A$ and the desired derivation can simply be:

$$\frac{\Gamma \vdash_{\leq} A}{\Gamma \vdash_{\leq} A \leq Top} \ (\mathsf{Top}\leq)$$

□

Observing that the rules defining well-formedness in $F_{\leq}$ are weaker than the ones in *F-bounded*, one can prove the following simple results.

Lemma 9.4 *Let Γ be a type environment and let A be a type;*

1. *if $\Gamma \vdash_{\leq} \diamondsuit$ then $\Gamma \vdash_{\flat} \diamondsuit$;*
2. *if $\Gamma \vdash_{\leq} A$ then $\Gamma \vdash_{\flat} A$.*

We now have all the necessary ingredients to prove the conservativity result for subtyping and typing.

Lemma 9.5 (conservativity of subtyping) *Let $\Gamma \vdash_{\leq} A$ and $\Gamma \vdash_{\leq} B$. Then:*

$$\Gamma \vdash_{\leq} A \leq B \qquad \textit{iff} \qquad \Gamma \vdash_{\flat} A \leq B$$

Proof.
($\Rightarrow$) We proceed by induction on the size of the derivation and by cases on the last rule applied. For cases ($\mathsf{IdVar} \leq$) and ($\mathsf{Top} \leq$), recall that these are (instances of) *F-bounded* rules and use Lemma 9.4. Similarly, for cases ($\mathsf{TVar} \leq$) and ($\rightarrow\leq$) use the fact that such rules are in *F-bounded* and apply the inductive hypothesis. Finally, if the last rule is ($\forall \leq_{\mathsf{sub}}$), the shape of the derivation is:

$$\frac{\Gamma \vdash_{\leq} A' \leq A \quad \Gamma, \alpha \leq A' \vdash_{\leq} B \leq B'}{\Gamma \vdash_{\leq} (\forall \alpha \leq A.\, B) \leq (\forall \alpha \leq A'.\, B')} \ (\forall \leq_{\mathsf{sub}})$$

By induction, $\Gamma \vdash_{\leq} A' \leq A$ implies $\Gamma \vdash_{\flat} A' \leq A$, and thus, by Weakening Lemma 4.10, $\Gamma, \alpha \leq A' \vdash_{\flat} A' \leq A$. Moreover, by induction $\Gamma, \alpha \leq A' \vdash_{\leq} B \leq B'$ implies $\Gamma, \alpha \leq A' \vdash_{\flat} B \leq B'$. The thesis follows by rule ($\forall' \leq$), which is provable in system *F-bounded* by Proposition 8.4:

$$\frac{\Gamma, \alpha \leq A' \vdash_{\flat} A' \leq A \quad \Gamma, \alpha \leq A' \vdash_{\flat} B \leq B'}{\Gamma \vdash_{\flat} \forall \alpha \leq A.\, B \leq \forall \alpha \leq A'.\, B'} \ (\forall' \leq)$$

($\Leftarrow$) It is convenient to consider the deterministic version *dF-bounded* of system *F-bounded*. Cases ($\mathsf{IdVar} \leq$) and ($\mathsf{Top} \leq$) are dealt with by the well-formedness hypothesis of $\Gamma \vdash A \leq B$ in $F_{\leq}$. Cases ($\mathsf{TVar} \leq$) and ($\rightarrow\leq$) are dealt with by induction. Finally, let the last rule be ($\forall \leq$):

$$\frac{\Gamma, \alpha \leq A' \vdash_{\flat} \alpha \leq A \quad \Gamma, \alpha \leq A' \vdash_{\flat} B \leq B'}{\Gamma \vdash_{\flat} \forall \alpha \leq A.\, B \leq \forall \alpha \leq A'.\, B'} \ (\forall \leq)$$

By induction, $\Gamma, \alpha \leq A' \vdash_{\flat} \alpha \leq A$ implies $\Gamma, \alpha \leq A' \vdash_{\leq} \alpha \leq A$, and thus, by Lemma 9.3, $\Gamma \vdash_{\leq} A' \leq A$. Notice that Lemma 9.3 can be applied since A is a bound for α, therefore, by definition of $F_{\leq}$ types, $\alpha \notin FV(A)$. By induction, $\Gamma, \alpha \leq A' \vdash_{\flat} B \leq B'$ implies $\Gamma, \alpha \leq A' \vdash_{\leq} B \leq B'$. Hence we can prove the thesis as follows:

$$\frac{\Gamma \vdash_{\leq} A' \leq A \quad \Gamma, \alpha \leq A' \vdash_{\leq} B \leq B'}{\Gamma \vdash_{\leq} (\forall \alpha \leq A.\, B) \leq (\forall \alpha \leq A'.\, B')} \ (\forall \leq_{\mathsf{sub}})$$

□

The conservativity of typing is now an easy corollary.

Theorem 9.6 (conservativity of typing) *Let $\Gamma, \Delta \vdash a : A$ be any well-formed* $\mathrm{F}_{\leq}$ *typing judgement. Then:*

$$\Gamma, \Delta \vdash_{\leq} a : A \quad \textit{iff} \quad \Gamma, \Delta \vdash_{\flat} a : A$$

Proof. Again it is convenient to consider the deterministic version *dF-bounded* of the system *F-bounded*. The basic remark is that, if $\forall \alpha \leq A.\, B$ and $\forall \alpha \leq A'.\, B'$ are *(Alg)*$F_{\leq}$ types, i.e., α does not occur (free) in the bounds A and A' thus the rule $(\mathsf{d}\forall\mathsf{E})$ of *dF-bounded* coincides with $(\forall\mathsf{E})_{\mathsf{sub}}$ in $AlgF_{\leq}$. This shows that a derivation in $AlgF_{\leq}$ is also a derivation in *dF-bounded* and thus proves ($\Rightarrow$).

As for ($\Leftarrow$), it suffices to notice that in each *F-bounded* rule, if the conclusion is a well-formed $F_{\leq}$ judgement (type variables do not appear in their bounds) then the judgements in the premises are well-formed as well. Then, an inductive reasoning that uses the above remark allows us to conclude. □

The undecidability of (sub)typing, proved in [Pie94] for system $F_{\leq}$, can now be extended to system *F-bounded*.

Corollary 9.7 *Subtyping is not decidable for system* F-bounded*.*

Proof. Subtyping is undecidable for system $F_{\leq}$, and, by Theorem 9.5, any algorithm for system *F-bounded* subtyping would also decide $F_{\leq}$ subtyping. □

We can also easily prove the non-conservativity of strong recursion for *F-bounded* subtyping, by extending a similar result given in [Ghe93] for system $F_{\leq}$. For the sake of brevity we only sketch the essential constructions. The interested reader can find more details in [Ghe93].

A common abstract notation for recursive types is $\mu X.A$, where X is a (recursion) type variable typically occurring in type A (recursive types are defined in most real languages via a construct of the form *let rec* $X = A$). We can distinguish two (families of) approaches to type level recursion, usually referred to as *weak recursion* and *strong recursion*. In the strong approach the type $\mu X.A$ is seen as the only solution of the equation $X = A$. Therefore the type equality $\mu X.A = A[X \leftarrow \mu X.A]$ holds in a "strong" sense (see [AC93, CG99]). The weak approach, on the other hand, only provides a couple of functions $\mathit{fold}_{\mu X.A}\colon A[X \leftarrow \mu X.A] \rightarrow \mu X.A$ and $\mathit{unfold}_{\mu X.A}\colon \mu X.A \rightarrow A[X \leftarrow \mu X.A]$ which allow the programmer to pass explicitly from a recursive type to its unfolding and vice versa [GMW79, AC96b]. The weak approach makes type and subtype checking very simple. The strong approach, instead, is easier for programmers to use, but makes subtype checking much more challenging; intermediate approaches are investigated in [Ghe93]. The non-conservativity result applies to strong recursion as well as to some intermediate approaches.

Let *μF-bounded* be any extension of system *F-bounded* with recursion variables named $X, Y, \ldots$ and with a constructor $\mu X.B$ for type recursion, such that the following rules are admissible (i.e., they express a deduction which can actually be proved in *μF-bounded*). Observe that such rules are admissible in any transitive

system with strong recursion, but they are actually weaker than strong recursion (see [AC93, Ghe93]).

$$\frac{\Gamma \vdash A[X{\leftarrow}\mu X.A] \leq B}{\Gamma \vdash \mu X.A \leq B}\ (\mathsf{unfold}-\mathsf{l}\leq) \qquad \frac{\Gamma \vdash B \leq A[X{\leftarrow}\mu X.A]}{\Gamma \vdash B \leq \mu X.A}\ (\mathsf{unfold}-\mathsf{r}\leq)$$

Consider now the following types, where $-A$ stands for $A \to \mathit{Top}$, and $\forall\alpha.\, A$ abbreviates $\forall\alpha \leq \mathit{Top}.\, A$.

$$\begin{array}{lcl} B & \equiv & \forall\alpha.\, -\forall\alpha' \leq \alpha.\, -\alpha \\ A & \equiv & \forall\beta \leq B.\, \beta \\ A' & \equiv & \forall\beta \leq B.\, \forall\beta' \leq \beta.\, -\beta \\ R & \equiv & \forall\beta \leq B.\, \mu X.\forall\beta' \leq X.\, -X \end{array}$$

The paper [Ghe93] shows that, in system $F_{\leq}$, the type A is not a subtype of A'; by Theorem 9.5 the same holds in system *F-bounded*. Now, in [Ghe93] it is also proved that both $\vdash A \leq R$ and $\vdash R \leq A'$ can be derived in any extension of system $F_{\leq}$ where the two unfold rules above are admissible, hence they are also derivable in any extension of system *F-bounded* where the same unfold rules are admissible. Therefore we obtain the following corollary.

Corollary 9.8 (non-conservativity of recursion) *There exist two types A and A' such that $\vdash A \leq A'$ does not hold in system* F-bounded*, while it holds in any extension of the system with a constructor $\mu X.B$ for type recursion and where the subtype relation is transitive and the two rules* $(\mathsf{unfold}-\mathsf{l}\leq)$ *and* $(\mathsf{unfold}-\mathsf{r}\leq)$ *are admissible.*

The paper [Ghe93] also contains a *limitation of non-conservativity* result. Let us say that $\Gamma \vdash A \leq A'$ is a non-conservative $F_{\leq}$ judgement if it does not contain recursive types, it does not hold in pure $F_{\leq}$, but it is derivable in the extended system obtained by adding recursion and the two unfold rules to $F_{\leq}$. The "limitation" result shows that every non-conservative $F_{\leq}$ judgement makes the standard subtype checking algorithm diverge; this is very interesting since we know from [Ghe95] that only "very special" judgements diverge. We conjecture that the same limitation result can be proved for system *F-bounded* too, but we leave this as an open problem.

10 PER semantics

The semantic interpretation that we propose for system *F-bounded* is obtained by adapting the semantics of system $F_{\leq}$, based on partial equivalence relations, first defined in [BL90] (see also [CL91, CMMS94, Ghe90]). Let $\langle\omega, .\rangle$ be Kleene's applicative structure, i.e., for $i, n \in \omega$, $i.n$ denotes the application of the i^{th} function in a Gödel numbering, to the argument n.[9] A *partial equivalence relation (p.e.r.)* p on ω is a transitive and symmetric relation on ω. A p.e.r. p can then be seen as an equivalence on the set $\{n \in \omega : n\ p\ n\}$ which is called its domain $\mathit{dom}(p)$. The quotient $\mathit{dom}(p)/_p$ is denoted by $\mathbf{Q}(p)$, namely $\mathbf{Q}(p) = \{[n]_p : n \in \mathit{dom}(p)\}$. We will often manipulate p.e.r.'s as sets of pairs, in particular by writing $p \subseteq q$ for $i\ p\ j \Rightarrow i\ q\ j$, and $p \cap q$ for $\{\langle i, j\rangle |\ i\ p\ j \wedge\ i\ q\ j\}$.

[9] Any other combinatory algebra would be appropriate.

In this approach a type A is interpreted as a p.e.r. $[\![A]\!]$. The idea is that the possible values of type A are the elements in $dom([\![A]\!])$ and if $i[\![A]\!]j$ then i, j represent values in $[\![A]\!]$ which cannot be discriminated by using the operations allowed on type A. Terms are then interpreted as equivalence classes. The interpretation is required to be sound with respect to the (sub)typing system, namely if $a : A$ is provable, then $[\![a : A]\!]$ must be a value in $[\![A]\!]$ (an equivalence class in $\mathbf{Q}([\![A]\!])$), and if $A \leq B$ then $[\![A]\!] \subseteq [\![B]\!]$, i.e., $i[\![A]\!]j \Rightarrow i[\![B]\!]j$. Notice that the inclusion between $[\![A]\!]$ and $[\![B]\!]$ expresses at the same time two basic intuitions about subtyping: the fact that every element of the subtype also belongs to the supertype (domain inclusion) and the fact that every function which can be used to discriminate elements of the supertype U can also be used to discriminate elements of the subtype T (namely, $\neg(i[\![B]\!]j) \Rightarrow \neg(i[\![A]\!]j)$).

To deal with free variables, we first interpret a judgement $\Gamma, \Delta \vdash \Diamond$ as the set of all well-typed assignments to the variables in Γ, Δ, and then we interpret a well-typed term $\Gamma, \Delta \vdash a : A$ by a function from $[\![\Gamma, \Delta \vdash \Diamond]\!]$ to $[\![\Gamma \vdash A]\!]$, i.e., a function which associates a value with any possible assignment of values to free variables. In the same way, a type $\Gamma \vdash A$ is interpreted by a function $[\![\Gamma \vdash A]\!]$ which associates a p.e.r. $[\![\Gamma \vdash A]\!]\gamma$ with each assignment γ of p.e.r.'s to the type variables. Specifically, $[\![\Gamma \vdash \mathit{Top}]\!]\gamma$ is always the total p.e.r. $\omega \times \omega$, which contains all values, but all of them are equivalent. To interpret arrow types, we first define the operator $(\ \rightarrow\ )$ on p.e.r.'s: if p, q are p.e.r.'s, then

$$i(p \rightarrow q)j \quad \Leftrightarrow \quad \forall m, n.\ m\ p\ n \Rightarrow i.m\ q\ j.n$$

i.e., two integers are related by $(p \rightarrow q)$ if they are the indexes of two functions which map p-related values to q-related values. Then, the interpretation $[\![\Gamma \vdash A \rightarrow B]\!]\gamma$ is simply defined as $([\![\Gamma \vdash A]\!]\gamma \rightarrow [\![\Gamma \vdash B]\!]\gamma)$. Finally, a universal type $\forall\alpha \leq A.\, B$ is interpreted as the intersection of all $[\![B[\alpha]]\!]$'s, when α ranges over all the p.e.r.'s such that $\alpha \subseteq [\![A[\alpha]]\!]$ (the formal definition is given later). The use of an intersection, rather than a function type, expresses the fact that the type parameter does not have any role in the computation, but is only used for type checking purposes. Hence, a term of type $\forall\alpha \leq A.\, B$ is not really interpreted as a function which takes a p.e.r. α and gives back a value in $[\![B[\alpha]]\!]$, but is just a constant value which is in every $[\![B[\alpha]]\!]$, regardless of what α really is. This essential property, which is at the base of most compilation techniques of polymorphic languages, is usually called "parametricity".

Notation 10.1 In the following, p.e.r's will be denoted by p and equivalence classes in $\mathbf{Q}(p)$ by v, possibly with subscripts. Finally, given a function $f : X \rightarrow Y$, $x_0 \in X$ and $y_0 \in Y$ we denote by $f[x_0 \mapsto y_0]$ the function from X to Y defined as $f[x_0 \mapsto y_0](x) = f(x)$ if $x \neq x_0$ and y_0 otherwise.

The following proposition introduces some properties of the $(\ \rightarrow\ )$ operator and of p.e.r.'s intersections which will be used hereafter.

Proposition 10.2 *1. Let p_1, p_2 be two p.e.r.'s; then the relation $(p_1 \rightarrow p_2)$ defined for all $i, j \in \omega$:*

$$i(p_1 \rightarrow p_2)j \qquad \textit{iff} \qquad \forall n, m.\ n\ p_1\ m \Rightarrow i.n\ p_2\ j.m,$$

is a p.e.r.; moreover, if $p_1' \subseteq p_1$ and $p_2 \subseteq p_2'$ then $(p_1 \rightarrow p_2) \subseteq (p_1' \rightarrow p_2')$.

2. *Let $\{p_i\}_{i\in I}$ be a collection of p.e.r.'s; the relation $\bigcap_{i\in I} p_i$ is a p.e.r., with*

   (a) $dom(\bigcap_{i\in I} p_i) = \bigcap_{i\in I} dom(p_i)$;

   (b) $\mathbf{Q}(\bigcap_{i\in I} p_i) = \{\bigcap_{i\in I} v_i : \{v_i\}_{i\in I} \in \Pi_{i\in I}\mathbf{Q}(p_i) \ \wedge \ \bigcap_{i\in I} v_i \neq \emptyset\}$.[10]

Since in case (2) the notation is a little complex, the reader could get some clearer ideas by considering the binary case, namely the intersection of two p.e.r.'s p_1 and p_2, which is a p.e.r. with domain $dom(p_1 \cap p_2) = dom(p_1) \cap dom(p_2)$ and classes $\mathbf{Q}(p_1 \cap p_2) = \{v_1 \cap v_2 :$ for all $\langle v_1, v_2\rangle \in \mathbf{Q}(p_1) \times \mathbf{Q}(p_2)$ such that $v_1 \cap v_2 \neq \emptyset\}$.

Semantics of types and environments

We are now ready to give the actual semantics. As discussed before, to give a semantics to a type or term containing free variables, we must specify a suitable semantic assignment to its variables. This is formalized by the notions of semantic type environment and value type environment. A *semantic type environment* is a function γ which associates a p.e.r. on ω with each type variable:

$$\gamma : \mathit{TypeVar} \to \mathit{PER},$$

where PER denotes the set of all p.e.r.'s on ω. A *semantic value environment* δ associates with each value variable a subset of ω to be interpreted as an equivalence class with respect to the p.e.r. denoted by the type of the variable:

$$\delta : \mathit{ValVar} \to \mathcal{P}(\omega)$$

Type judgements are interpreted as functions which, given a semantic environment γ, return the p.e.r. denoted by the type, where free variables are interpreted according to γ.

$$\begin{array}{lcl}
[\![\Gamma \vdash \mathit{Top}]\!]\gamma & = & \omega \times \omega \\
[\![\Gamma', \alpha \leq A, \Gamma'' \vdash \alpha]\!]\gamma & = & \gamma(\alpha) \\
[\![\Gamma \vdash A \to B]\!]\gamma & = & ([\![\Gamma \vdash A]\!]\gamma \to [\![\Gamma \vdash B]\!]\gamma) \\
[\![\Gamma \vdash \forall\alpha \leq A.\, B]\!]\gamma & = & \bigcap_{p \subseteq [\![\Gamma, \alpha \leq A \vdash A]\!]\gamma[\alpha \mapsto p]} [\![\Gamma, \alpha \leq A \vdash B]\!]\gamma[\alpha \mapsto p]
\end{array}$$

Since the semantics of $\Gamma \vdash A$ does not depend on Γ, we will often write $[\![A]\!]\gamma$ for $[\![\Gamma \vdash A]\!]\gamma$.

We say that a semantic environment γ satisfies a (syntactic) environment Γ if the assignment to the variables in Γ are consistent with the constraints imposed by type bounds on type variables and typing on value variables. First, the notion of semantic type environment γ satisfying a (syntactic) environment Γ, written $\gamma \models \Gamma$, is defined inductively as follows:

$$\begin{array}{ll}
\gamma \models \epsilon & \\
\gamma \models \Gamma, \alpha \leq A & \text{if } \gamma \models \Gamma \text{ and } \gamma(\alpha) \subseteq [\![A]\!]\gamma
\end{array}$$

Given $\gamma \models \Gamma$, the notion of a semantic value environment satisfying Γ, Δ, written $\gamma, \delta \models \Gamma, \Delta$ is defined inductively as follows:

$$\begin{array}{ll}
\gamma, \delta \models \Gamma, \epsilon & \text{if } \gamma \models \Gamma \\
\gamma, \delta \models \Gamma, \Delta, x\!:\!A & \text{if } \gamma, \delta \models \Gamma, \Delta \text{ and } \delta(x) \in \mathbf{Q}([\![A]\!]\gamma).
\end{array}$$

[10] The notation $\{v_i\}_{i\in I} \in \Pi_{i\in I}\mathbf{Q}(p_i)$ means that $\forall i \in I.\ v_i \in \mathbf{Q}(p_i)$.

Syntactic environments are interpreted by the sets of semantic environments which represent well-typed assignments to all the type and value variables in the environment (to simplify the notation we write $[\![\Gamma, \Delta]\!]$ instead of $[\![\Gamma, \Delta \vdash \diamondsuit]\!]$). Therefore

$$[\![\Gamma]\!] = \{\gamma : \gamma \models \Gamma\}$$
$$[\![\Gamma, \Delta]\!] = \{\langle\gamma, \delta\rangle : \gamma, \delta \models \Gamma, \Delta\}$$

Notice that the semantics of $\Gamma, \alpha \leq A$ depends on the semantics of the type A where α may occur free, namely, expanding the notation, on the semantics of a judgement $\Gamma, \alpha \leq A \vdash A$. However, the fact that the environment $\Gamma, \alpha \leq A$ appears in this judgement does not create any circularity in the definition, since as already noticed, the interpretation of types does not depend on (the interpretation of) environments.

Semantics of terms

The interpretation of well-typed terms is given by induction on the typing derivation, and by cases on the last rule applied. For this reason, we should use a notation like $[\![d]\!]\langle\gamma, \delta\rangle$, where d is a notation for a typing derivation. However, to keep things simple, we do not write the full derivation as the argument of the semantic function but just the proved judgement, and in the next definition we assume that the predecessors of the final judgement are the same as in the presentation of Section 3. We will later prove a coherence theorem which states that indeed our interpretation only depends on the proved judgement, hence justifying the notation.

Notice that, in cases (Subs), ($\rightarrow$ E) and ($\forall$E), the interpretation is built as $\bigcup [i]_{[\![B]\!]\gamma}$, where B is the type of the term and i ranges over a suitable set of integers. The idea is that all the i's should belong to the same equivalence class and thus it would be sufficient to take the equivalence class of only one of them; but this fact will be proved only later, in Theorem 10.11. This result will also imply that in cases (Subs), ($\forall$I) and ($\forall$E) the interpretation can be obtained by choosing any element i in the equivalence class interpreting the main premise, and changing only the p.e.r. where its equivalence class is considered. This fact has an interesting practical interpretation: if every term is compiled to an index in its equivalence class, then no code needs to be generated for subsumption, second order abstraction and second order application. This is what usually happens in actual implementations.

Definition 10.3 A typing derivation is interpreted by a subset of ω defined as follows.

$$
\begin{array}{ll}
(\mathsf{Var}) & [\![\Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i]\!]\langle\gamma,\delta\rangle \\
& = \quad \delta(x_i) \\
(\mathsf{Subs}) & [\![\Gamma, \Delta \vdash a : B]\!]\langle\gamma,\delta\rangle \\
& = \quad \bigcup [i]_{[\![B]\!]\gamma}, \text{ for } i \in [\![\Gamma, \Delta \vdash a : A]\!]\langle\gamma,\delta\rangle \\
(\to \mathsf{I}) & [\![\Gamma, \Delta \vdash \lambda x{:}A.\, b : A \to B]\!]\langle\gamma,\delta\rangle \\
& = \quad \{i \in \omega \mid \forall v \in \mathbf{Q}([\![A]\!]\gamma),\ \forall j \in v,\ i.j \in [\![\Gamma, \Delta, x : A \vdash b : B]\!]\langle\gamma, \delta[x \mapsto v]\rangle\} \\
(\to \mathsf{E}) & [\![\Gamma, \Delta \vdash f(a) : B]\!]\langle\gamma,\delta\rangle \\
& = \quad \bigcup [i.j]_{[\![B]\!]\gamma},\ \textit{for } i \in [\![\Gamma, \Delta \vdash f : A \to B]\!]\langle\gamma,\delta\rangle, j \in [\![\Gamma, \Delta \vdash a : A]\!]\langle\gamma,\delta\rangle \\
(\forall \mathsf{I}) & [\![\Gamma, \Delta \vdash \Lambda\alpha \leq A.\, b : \forall \alpha \leq A.\, B]\!]\langle\gamma,\delta\rangle \\
& = \quad \bigcap_{p \subseteq [\![A]\!]\gamma[\alpha \leftarrow p]} [\![\Gamma, \alpha \leq A, \Delta \vdash b : B]\!]\langle\gamma[\alpha \mapsto p], \delta\rangle \\
(\forall \mathsf{E}) & [\![\Gamma, \Delta \vdash f\{A'\} : B[\alpha \leftarrow A']]\!]\langle\gamma,\delta\rangle \\
& = \quad \bigcup [i]_{[\![B[\alpha \leftarrow A']]\!]\gamma},\ \textit{for } i \in [\![\Gamma, \Delta \vdash f : \forall \alpha \leq A.\, B]\!]\langle\gamma,\delta\rangle
\end{array}
$$

It is not difficult to prove that the above semantics is well-defined.

Theorem 10.4 (definition) *If $\Gamma \vdash A$ then $[\![A]\!]\gamma$ is a uniquely defined p.e.r. If $\Gamma \vdash \diamondsuit$, then $[\![\Gamma]\!]$ is a uniquely defined semantic type environment. If $\Gamma, \Delta \vdash \diamondsuit$, then $[\![\Gamma, \Delta]\!]$ is a uniquely defined semantic value environment. If d proves $\Gamma, \Delta \vdash a : A$, and $\langle\gamma,\delta\rangle \in [\![\Gamma, \Delta]\!]$ then $[\![d]\!]\langle\gamma,\delta\rangle$ is a uniquely defined subset of ω.*

Proof. For types, the fact that $[\![A]\!]\gamma$ is well-defined for any semantic type environment γ can easily be proved by using Proposition 10.2. In particular notice that the intersection of a set of p.e.r.'s is a p.e.r., and that at least the empty p.e.r. satisfies the condition $p \subseteq [\![\Gamma, \alpha \leq A \vdash A]\!]\gamma[\alpha \mapsto p]$. For environments, no doubts should arise. For derivations, the semantics has been defined in such a way that it is always a well-defined set of integers by construction. The price to pay for this is that, in principle, it is not obvious that this set is not empty, and that it is an equivalence class of the corresponding type (Theorem 10.11). □

The first basic property enjoyed by the proposed semantics is the soundness of subtyping, namely the fact the subtyping relation on types has set-theoretical inclusion as a semantical counterpart.

Theorem 10.5 (soundness of subtyping) *If $\Gamma \vdash A \leq B$, then, $\forall \gamma \in [\![\Gamma]\!]$, we have $[\![A]\!]\gamma \subseteq [\![B]\!]\gamma$.*

Proof. By induction on the structure of the derivation of $\Gamma \vdash A \leq B$ and by cases on the last rule applied. In the cases ($\mathsf{Id} \leq$) and ($\mathsf{Trans} \leq$) we simply use reflexivity and transitivity of subset inclusion. For ($\mathsf{Top} \leq$) just notice that $[\![Top]\!]\gamma = \omega \times \omega$ is the greatest p.e.r.. The case ($\mathsf{Var} \leq$) follows directly from the definition of the semantics of environments and for ($\to\leq$) we use the property of the function space operator stated in Proposition 10.2(1).

The interesting case is rule ($\forall \leq$). Suppose that the last rule applied in the derivation is

$$\frac{\Gamma, \alpha \leq A' \vdash \alpha \leq A \quad \Gamma, \alpha \leq A' \vdash B \leq B'}{\Gamma \vdash (\forall \alpha \leq A.\, B) \leq (\forall \alpha \leq A'.\, B')}\ (\forall \leq)$$

Let $\gamma \in [\![\Gamma]\!]$; we have to prove that

$$\bigcap_{p\subseteq[\![A]\!]\gamma[\alpha\mapsto p]}[\![B]\!]\gamma[\alpha\mapsto p] \subseteq \bigcap_{p\subseteq[\![A']\!]\gamma[\alpha\mapsto p]}[\![B']\!]\gamma[\alpha\mapsto p].$$

By inductive hypothesis we know that:

1. $p \subseteq [\![A']\!]\gamma[\alpha\mapsto p] \;\Rightarrow\; p \subseteq [\![A]\!]\gamma[\alpha\mapsto p]$
2. $p \subseteq [\![A']\!]\gamma[\alpha\mapsto p] \;\Rightarrow\; [\![B]\!]\gamma[\alpha\mapsto p] \subseteq [\![B']\!]\gamma[\alpha\mapsto p]$

From 1, 2 we deduce 1, 2 below, and thus we conclude by transitivity of subset inclusion.

1. $\bigcap_{p\subseteq[\![A]\!]\gamma[\alpha\mapsto p]}[\![B]\!]\gamma[\alpha\mapsto p] \subseteq \bigcap_{p\subseteq[\![A']\!]\gamma[\alpha\mapsto p]}[\![B]\!]\gamma[\alpha\mapsto p]$
2. $\bigcap_{p\subseteq[\![A']\!]\gamma[\alpha\mapsto p]}[\![B]\!]\gamma[\alpha\mapsto p] \subseteq \bigcap_{p\subseteq[\![A']\!]\gamma[\alpha\mapsto p]}[\![B']\!]\gamma[\alpha\mapsto p]$ □

We next introduce untyped lambda terms and we interpret in the obvious way each untyped term (in a given variable environment) with a computable function. Then we show that the meaning of a typed term can be nicely characterized by using the function associated with its erasure. Such a result will allow us to easily conclude the soundness of typing and coherence results.

Definition 10.6 ***(untyped λ-terms)*** The set Λ of *untyped lambda terms* is defined by the following grammar, where x denotes a generic value variable:

$$U \;::=\; x \mid U(U) \mid \lambda x.\, U$$

Untyped terms will be denoted by u, possibly with subscripts.

Definition 10.7 ***(erasure)*** Let a be an *F-bounded* term. The *erasure* of a is the (untyped) term $\mathit{erase}(a) \in \Lambda$ defined as follows:

$$\begin{array}{lcl}
\mathit{erase}(x) & = & x \\
\mathit{erase}(\lambda x{:}A.\, b) & = & \lambda x.\, \mathit{erase}(b) \\
\mathit{erase}(f(a)) & = & \mathit{erase}(f)(\mathit{erase}(a)) \\
\mathit{erase}(\Lambda\alpha{\leq}A.\, b) & = & \mathit{erase}(b) \\
\mathit{erase}(b\{A\}) & = & \mathit{erase}(b)
\end{array}$$

Definition 10.8 For any untyped term $u \in \Lambda$ and variables $x_1, \ldots, x_n$, such that $FV(u) \subseteq \{x_1, \ldots, x_n\}$, we define a function $F_u^{x_1,\ldots,x_n} : \mathbb{N}^n \to \mathbb{N}$ as follows: for all $i_1, \ldots, i_n \in \mathbb{N}$:

$$\begin{array}{lcl}
F_{x_k}^{x_1,\ldots,x_n}(i_1,\ldots,i_n) & = & i_k \\
F_{\lambda x_{n+1}.\, u}^{x_1,\ldots,x_n}(i_1,\ldots,i_n) & = & \text{a Gödel index for the function} \\
 & & \qquad i_{n+1} \mapsto F_u^{x_1,\ldots,x_n,x_{n+1}}(i_1,\ldots,i_n,i_{n+1}) \\
F_{u_1(u_2)}^{x_1,\ldots,x_n}(i_1,\ldots,i_n) & = & F_{u_1}^{x_1,\ldots,x_n}(i_1,\ldots,i_n).F_{u_2}^{x_1,\ldots,x_n}(i_1,\ldots,i_n)
\end{array}$$

One can easily see that each $F_u^{x_1,\ldots,x_n}$ is well-defined[11] and computable. This can be proved inductively, by observing that in the first clause we just define the projection

[11] Not uniquely, due to the existence of (infinitely) many indexes for the same computable function.

on the k^{th} component, by using the *s-m-n theorem* from computability for the second clause and the existence of a universal computable function for the third one.[12]

A simple technical lemma, regarding the effect of substitution at a semantic level for types and terms will be needed in the following.

Lemma 10.9 (semantic substitution) *1. Let* $\Gamma, \alpha \leq A, \Gamma' \vdash B$ *and* $\Gamma, \Gamma'[\alpha \leftarrow A'] \vdash A'$. *Then, for any semantic type environment* γ *we have*

$$[\![B]\!]\gamma[\alpha \mapsto [\![A']\!]\gamma] = [\![B[\alpha \leftarrow A']]\!]\gamma$$

2. Let $\Gamma, \Delta, x : A, \Delta' \vdash b : B$ *and* $\Gamma, \Delta, \Delta' \vdash a : A$. *Then, for any* $\langle \gamma, \delta \rangle \in [\![\Gamma, \Delta, \Delta']\!]$ *we have:*

$$[\![\Gamma, \Delta, \Delta' \vdash b[x \leftarrow a] : B]\!]\langle \gamma, \delta \rangle = [\![\Gamma, \Delta, x : A, \Delta' \vdash b : B]\!]\langle \gamma, \delta[x \mapsto v] \rangle,$$

where $v = [\![\Gamma, \Delta, \Delta' \vdash a : A]\!]\langle \gamma, \delta \rangle$

Proof. Both points are proved by straightforward induction (on the structure of the type B and of the term b, respectively). □

The next result essentially asserts that the interpretation of typed terms can be obtained from the above interpretation of untyped terms, by taking the quotient with respect to the corresponding type. It immediately implies the soundness of typing and the coherence result for the semantics.

Lemma 10.10 *Let* d *be a derivation of* $\Gamma, \Delta \vdash a : A$ *in* F-bounded, *where* $\Delta \equiv x_1 : A_1, \ldots, x_n : A_n$, *and let* $\langle \gamma, \delta \rangle \in [\![\Gamma, \Delta]\!]$. *Then choosing* $i_k \in \delta(x_k)$ *for* $k \in \{1, \ldots, n\}$, *we have*

$$[\![d]\!]\langle \gamma, \delta \rangle = [F^{x_1,\ldots,x_n}_{erase(a)}(i_1, \ldots, i_n)]_{[\![A]\!]\gamma}.$$

Proof. Let $\langle \gamma, \delta \rangle \in [\![\Gamma, \Delta]\!]$ and let $i_k \in \delta(x_k)$ for $k \in \{1, \ldots, n\}$. The proof proceeds by induction on the structure of the derivation d and by cases according to the last rule used in the derivation d. As usual, we do not indicate the entire derivation as the argument of the semantic function $[\![\cdot]\!]$ but only the proved judgement.

- (Var) Let the last rule be:

$$\frac{\Gamma, \Delta \vdash \Diamond}{\Gamma, \Delta \vdash x_k : A_k}\ (\mathsf{Var})$$

where Δ is $\Delta', x_k : A_k, \Delta''$. Then, by definition of the semantics of environments, $\delta(x_k) \in \mathbf{Q}([\![A_k]\!]\gamma)$ and, since $i_k \in \delta(x_k)$, we have $\delta(x_k) = [i_k]_{[\![A_k]\!]\gamma}$. Hence

$$\begin{aligned}
&[\![\Gamma, \Delta \vdash x_k : A_k]\!]\langle \gamma, \delta \rangle \\
&\quad = \delta(x_k) \\
&\quad = [i_k]_{[\![A_k]\!]\gamma} \\
&\quad = [F^{x_1,\ldots,x_n}_{x_k}(i_1, \ldots, i_n)]_{[\![A_k]\!]\gamma}
\end{aligned}$$

[12] Kleene application is intended to be undefined when one of the two arguments is undefined, and thus, if $\Psi_U : \mathbb{N}^2 \to \mathbb{N}$ is the universal function then $F^{x_1,\ldots,x_n}_{u_1}(i_1, \ldots, i_n).F^{x_1,\ldots,x_n}_{u_2}(i_1, \ldots, i_n)$ is $\Psi_U(F^{x_1,\ldots,x_n}_{u_1}(i_1, \ldots, i_n), F^{x_1,\ldots,x_n}_{u_2}(i_1, \ldots, i_n))$.

$F^{x_1,\ldots,x_n}_{x_k}$ being the projection on the k^{th} argument. Recalling that $erase(x_k) = x_k$ we can conclude.

- (Subs) Let the last rule be:

$$\frac{\Gamma, \Delta \vdash a : A \quad \Gamma \vdash A \leq B}{\Gamma, \Delta \vdash a : B} \ (\mathsf{Subs})$$

By soundness of subtyping (Lemma 10.5), since $\Gamma \vdash A \leq B$,

$$[\![A]\!]\gamma \subseteq [\![B]\!]\gamma. \tag{1}$$

Moreover, by induction hypothesis,

$$[\![\Gamma, \Delta \vdash a : A]\!]\langle\gamma, \delta\rangle = [F^{x_1,\ldots,x_n}_{erase(a)}(i_1, \ldots, i_n)]_{[\![A]\!]\gamma}. \tag{2}$$

Therefore

$$\begin{aligned}
&[\![\Gamma, \Delta \vdash a : B]\!]\langle\gamma, \delta\rangle \\
&\quad = \textstyle\bigcup_{i \in [\![\Gamma, \Delta \vdash a:A]\!]\langle\gamma,\delta\rangle} [i]_{[\![B]\!]\gamma} \\
&\quad = [i]_{[\![B]\!]\gamma} && \text{for any } i \in [\![\Gamma, \Delta \vdash a : A]\!]\langle\gamma, \delta\rangle \\
&\quad = [F^{x_1,\ldots,x_n}_{erase(a)}(i_1, \ldots, i_n)]_{[\![B]\!]\gamma} && \text{[by (1) and (2)]}
\end{aligned}$$

The last step uses the fact that $F^{x_1,\ldots,x_n}_{erase(a)}(i_1, \ldots, i_n) \in [\![\Gamma, \Delta \vdash a : A]\!]\langle\gamma, \delta\rangle$, by (2).

- ($\to$ I) Let the last rule be:

$$\frac{\Gamma, \Delta, x{:}A \vdash b : B}{\Gamma, \Delta \vdash \lambda x{:}A.\, b : A \to B} \ (\to \mathsf{I})$$

For any $v \in \mathbf{Q}([\![A]\!]\gamma)$, by definition of the semantics of environments, $\langle\gamma, \delta[x \mapsto v]\rangle \in [\![\Gamma, \Delta, x{:}A]\!]$. Therefore, by inductive hypothesis, choosing any $i \in v$,

$$[\![\Gamma, \Delta, x{:}A \vdash b : B]\!]\langle\gamma, \delta[x \mapsto v]\rangle = [F^{x_1,\ldots,x_n,x}_{erase(b)}(i_1, \ldots, i_n, i)]_{[\![B]\!]\gamma}$$

Now, by definition of $F^{x_1,\ldots,x_n}_u$, for any $v \in \mathbf{Q}([\![A]\!]\gamma)$ and $i \in v$, if we define $i_\lambda = F^{x_1,\ldots,x_n}_{\lambda x.\ erase(b)}(i_1, \ldots, i_n)$, we have that $i_\lambda.i = F^{x_1,\ldots,x_n,x}_{erase(b)}(i_1, \ldots, i_n, i)$. Therefore recalling the definition of the semantics of λ-abstraction

$$i_\lambda \in [\![\Gamma, \Delta \vdash \lambda x{:}A.\, b : A \to B]\!]\langle\gamma, \delta\rangle$$

By definition of $[\![A \to B]\!]\gamma$, any other index $j \in [i_\lambda]_{[\![A\to B]\!]\gamma}$ is in the semantics of the abstraction, and vice versa. Thus we conclude

$$[\![\Gamma, \Delta \vdash \lambda x{:}A.\, b : A \to B]\!]\langle\gamma, \delta\rangle = [i_\lambda]_{[\![A\to B]\!]\gamma},$$

which is the desired result, since $i_\lambda = F^{x_1,\ldots,x_n}_{\lambda x.\ erase(b)}(i_1, \ldots, i_n)$ and $erase(\lambda x{:}A.\, b) = \lambda x.\, erase(b)$.

- ($\to$ E) Let the last rule be:

$$\frac{\Gamma, \Delta \vdash f : A \to B \quad \Gamma, \Delta \vdash a : A}{\Gamma, \Delta \vdash f(a) : B} \ (\to \mathsf{E})$$

By inductive hypothesis, if we define $i_f = F^{x_1,\ldots,x_n}_{erase(f)}(i_1, \ldots, i_n)$ and $i_a = F^{x_1,\ldots,x_n}_{erase(a)}(i_1, \ldots, i_n)$, we have that

$$[\![\Gamma, \Delta \vdash f : A \to B]\!]\langle\gamma, \delta\rangle = [i_f]_{[\![A\to B]\!]\gamma} \quad \text{and} \quad [\![\Gamma, \Delta \vdash a : A]\!]\langle\gamma, \delta\rangle = [i_a]_{[\![A]\!]\gamma}$$

Now, since $i_f \in dom([\![A \to B]\!]\gamma)$, and $i_a \in dom([\![A]\!]\gamma)$, by the definition of $[\![A \to B]\!]\gamma$ we have that

$$i_f.i_a \in dom([\![B]\!]\gamma).$$

Moreover, exploiting the inductive hypothesis and the fact that $[\![A \to B]\!]\gamma = ([\![A]\!]\gamma \to [\![B]\!]\gamma)$, we have that, for any other $i'_f \in [\![\Gamma, \Delta \vdash f : A \to B]\!]\langle\gamma, \delta\rangle$ and $i'_a \in [\![\Gamma, \Delta \vdash a : A]\!]\langle\gamma, \delta\rangle$, $i'_f.i'_a([\![B]\!]\gamma)i_f.i_a$. Therefore:

$$\begin{aligned}
&[\![\Gamma, \Delta \vdash f(a) : B]\!]\langle\gamma, \delta\rangle \\
&\quad = \bigcup [i'_f.i'_a]_{[\![B]\!]\gamma}, \quad \begin{array}{l} \textit{for } i'_f \in [\![\Gamma, \Delta \vdash f : A \to B]\!]\langle\gamma, \delta\rangle \\ \textit{and } i'_a \in [\![\Gamma, \Delta \vdash a : A]\!]\langle\gamma, \delta\rangle \end{array} \\
&\quad = [i_f.i_a]_{[\![B]\!]\gamma} \\
&\quad = [F^{x_1,\ldots,x_n}_{erase(f)}(i_1, \ldots, i_n).F^{x_1,\ldots,x_n}_{erase(a)}(i_1, \ldots, i_n)]_{[\![B]\!]\gamma} \\
&\quad = [F^{x_1,\ldots,x_n}_{erase(f)(erase(a))}(i_1, \ldots, i_n)]_{[\![B]\!]\gamma},
\end{aligned}$$

that is what we want, since $erase(f(a)) = erase(f)(erase(a))$.

- ($\forall$I) Let the last rule be:

$$\frac{\Gamma, \alpha \leq A, \Delta \vdash b : B \quad \alpha \notin FV\Delta}{\Gamma, \Delta \vdash \Lambda\alpha \leq A.\, b : \forall\alpha \leq A.\, B} \ (\forall \mathsf{I})$$

Let p be any p.e.r. such that $p \subseteq [\![A]\!]\gamma[\alpha \mapsto p]$ and thus $\gamma[\alpha \mapsto p] \models \Gamma, \alpha \leq A$. Then, by inductive hypothesis, if we denote with $i_b = F^{x_1,\ldots,x_n}_{erase(b)}(i_1, \ldots, i_n)$[13], we have that

$$[\![\Gamma, \alpha \leq A, \Delta \vdash b : B]\!]\langle\gamma[\alpha \mapsto p], \delta\rangle = [i_b]_{[\![B]\!]\gamma[\alpha\mapsto p]}$$

Therefore, by definition of the semantics of terms we have

$$\begin{aligned}
&[\![\Gamma, \Delta \vdash \Lambda\alpha \leq A.\, b : \forall\alpha \leq A.\, B]\!]\langle\gamma, \delta\rangle \\
&\quad = \textstyle\bigcap_{p \subseteq [\![A]\!]\gamma[\alpha\mapsto p]} [\![\Gamma, \alpha \leq A, \Delta \vdash b : B]\!]\langle\gamma[\alpha \mapsto p], \delta\rangle \\
&\quad = \textstyle\bigcap_{p \subseteq [\![A]\!]\gamma[\alpha\mapsto p]} [i_b]_{[\![B]\!]\gamma[\alpha\mapsto p]}
\end{aligned}$$

[13] Notice that i_b is independent from p.

Recalling the definition of $[\![\forall\alpha\leq A.\,B]\!]\gamma$ and exploiting the fact that equivalence classes of a p.e.r. obtained as the intersection of a family of p.e.r.'s are the (non-empty) intersections of classes of the original p.e.r.'s (see Proposition 10.2(2)), we conclude from the above that

$$[\![\Gamma,\Delta\vdash\Lambda\alpha\leq A.\,b:\forall\alpha\leq A.\,B]\!]\langle\gamma,\delta\rangle=[i_b]_{[\![\forall\alpha\leq A.\,B]\!]\gamma}$$

which is exactly the desired result since $i_b = F^{x_1,\ldots,x_n}_{erase(b)}(i_1,\ldots,i_n)$ and $erase(b) = erase(\Lambda\alpha\leq A.\,b)$.

- ($\forall\mathsf{E}$) Let the last rule be:

$$\frac{\Gamma,\Delta\vdash f:\forall\alpha\leq A.\,B\quad\Gamma\vdash A'\leq A[\alpha\leftarrow A']}{\Gamma,\Delta\vdash f\{A'\}:B[\alpha\leftarrow A']}\ (\forall\mathsf{E})$$

Then, by inductive hypothesis, if we denote with $i_f = F^{x_1,\ldots,x_n}_{erase(f)}(i_1,\ldots,i_n)$, we have that

$$[\![\Gamma,\Delta\vdash f:\forall\alpha\leq A.\,B]\!]\langle\gamma,\delta\rangle=[i_f]_{[\![\forall\alpha\leq A.\,B]\!]\gamma}$$

Moreover, by soundness of subtyping (Lemma 10.5) and semantic substitution (Lemma 10.9(1)) we have that

$$[\![A']\!]\gamma\subseteq[\![A[\alpha\leftarrow A']]\!]\gamma=[\![A]\!]\gamma[\alpha\mapsto[\![A']\!]\gamma]$$

Since $[\![A']\!]\gamma$ satisfies the condition $p\subseteq[\![A]\!]\gamma[\alpha\mapsto p]$, by definition of the semantics of $\forall$-types,

$$\begin{aligned}&[\![\forall\alpha\leq A.\,B]\!]\gamma=\\&=\textstyle\bigcap_{p\subseteq[\![A]\!]\gamma[\alpha\mapsto p]}[\![B]\!]\gamma[\alpha\mapsto p]\\&\subseteq[\![B]\!]\gamma[\alpha\mapsto[\![A']\!]\gamma]\\&=[\![B[\alpha\leftarrow A']]\!]\gamma\qquad\qquad\text{[by Lemma 10.9(1)]}\end{aligned}$$

Therefore, noticing that $[\![\forall\alpha\leq A.\,B]\!]\gamma$ is a subset of $[\![B[\alpha\leftarrow A']]\!]\gamma$ and reasoning as in the case (Subs), we can conclude

$$[\![\Gamma,\Delta\vdash f\{A'\}:B[\alpha\leftarrow A']]\!]\langle\gamma,\delta\rangle=[i_f]_{[\![B[\alpha\leftarrow A']]\!]\gamma},$$

which is what we want, since $i_f = F^{x_1,\ldots,x_n}_{erase(f)}(i_1,\ldots,i_n)$ and $erase(f\{A\}) = erase(f)$. □

It is worth noticing that we could have defined directly the meaning of an *F-bounded* term by using the interpretation of its erasure and the semantics of types. This approach has been widely explored in the literature. The interested reader can consult the book [Gun92], where it is shown how a p.e.r. model of the second order polymorphic lambda calculus can be defined starting from a generic (untyped) lambda model. An explicit construction of a semantics for a variant of system $F_{\leq}$ is also carried out in [HP96].

The previous lemma immediately implies that a term a of type A is interpreted as an equivalence class (value) in the semantics of A. Such a result expresses the soundness of typing with respect to the semantics.

Corollary 10.11 (soundness of typing) *If* $\Gamma, \Delta \vdash a : A$*, then,* $\forall \langle \gamma, \delta \rangle \in [\![\Gamma, \Delta]\!]$*,* $[\![\Gamma, \Delta \vdash a : A]\!]\langle \gamma, \delta \rangle \in \mathbf{Q}([\![A]\!]\gamma)$*.*

Another immediate corollary states the non-emptiness of the interpretation of terms, namely the fact that the semantics of each well-typed term is non empty for each possible choice of the semantic environment. However it is worth noticing that the semantics of environments can be empty, as one can verify considering, for instance, the environment Γ, Δ with $\Gamma \equiv \epsilon$ and $\Delta \equiv x : \forall \alpha \leq \alpha.\, \alpha$.

Corollary 10.12 (non emptiness) *If* $\Gamma, \Delta \vdash a : A$ *and* $\langle \gamma, \delta \rangle \in [\![\Gamma, \Delta]\!]$ *then* $[\![\Gamma, \Delta \vdash a : A]\!]\langle \gamma, \delta \rangle \neq \emptyset$*.*

A last corollary expresses the fact that the semantics does not depend on the structure of the derivation of a judgement, but only on the judgement itself, a property known as the *coherence* of the semantics.

Theorem 10.13 (coherence) *If* d *and* d' *both prove the judgement* $\Gamma, \Delta \vdash a : A$*, and* $\langle \gamma, \delta \rangle \in [\![\Gamma, \Delta]\!]$*, then* $[\![d]\!]\langle \gamma, \delta \rangle = [\![d']\!]\langle \gamma, \delta \rangle$*.*

Proof. Just notice that $F^{x_1,\ldots,x_n}_{erase(a)}$ does not depend on the typing derivation and use Lemma 10.10. □

Remark 10.14 By Corollary 10.11, the interpretation of terms may be equivalently restated in the following simplified way.

(Var) $[\![\Gamma, x_1 : A_1, \ldots, x_n : A_n \vdash x_i : A_i]\!]\langle \gamma, \delta \rangle = [j]_{[\![A_i]\!]\gamma}$,
for any $j \in \delta(x_i)$

(Subs) $[\![\Gamma, \Delta \vdash a : B]\!]\langle \gamma, \delta \rangle = [i]_{[\![B]\!]\gamma}$,
for any $i \in [\![\Gamma, \Delta \vdash a : A]\!]\langle \gamma, \delta \rangle$

($\to$ I) $[\![\Gamma, \Delta \vdash \lambda x{:}A.\, b : A \to B]\!]\langle \gamma, \delta \rangle = [i]_{[\![A \to B]\!]\gamma}$,
for any $i \in \omega$ s.t. $\forall v \in \mathbf{Q}([\![A]\!]\gamma),\ \forall j \in v,\ i.j \in [\![\Gamma, \Delta, x : A \vdash b : B]\!]\langle \gamma, \delta[x \mapsto v] \rangle$

($\to$ E) $[\![\Gamma, \Delta \vdash f(a) : B]\!]\langle \gamma, \delta \rangle = [i.j]_{[\![B]\!]\gamma}$,
for any $i \in [\![\Gamma, \Delta \vdash f : A \to B]\!]\langle \gamma, \delta \rangle, j \in [\![\Gamma, \Delta \vdash a : A]\!]\langle \gamma, \delta \rangle$

($\forall$I) $[\![\Gamma, \Delta \vdash \Lambda \alpha \leq A.\, b : \forall \alpha \leq A.\, B]\!]\langle \gamma, \delta \rangle = [i]_{[\![\forall \alpha \leq A.\, B]\!]\gamma}$,
for any $p \subseteq [\![A]\!]\gamma[\alpha \mapsto p], i \in [\![\Gamma, \alpha \leq A, \Delta \vdash b : B]\!]\langle \gamma[\alpha \mapsto p], \delta \rangle$

($\forall$E) $[\![\Gamma, \Delta \vdash f\{A'\} : B[\alpha \leftarrow A']]\!]\langle \gamma, \delta \rangle = [i]_{[\![B[\alpha \leftarrow A']]\!]\gamma}$,
for any $i \in [\![\Gamma, \Delta \vdash f : \forall \alpha \leq A.\, B]\!]\langle \gamma, \delta \rangle$

Equational system

Finally, we introduce an equational system for judgements of the shape $\Gamma, \Delta \vdash a = b : A$, meaning that terms a and b represent indistinguishable elements of type A, when free type and value variables are instantiated consistently with the constraints specified by the environments Γ and Δ, respectively. The equational system is then formally proved to be sound with respect to the semantics. The rules of the system, listed in Table 2, are essentially the same as those for system $F_{\leq}$, namely:

- type and term versions of β and η rules;
- *reflexivity*, *symmetry* and *transitivity* to obtain an equivalence;
- *structural rules* to force the equivalence to be a congruence;
- a *"top" rule* which states that all terms are indistinguishable in the *Top* type (as in [Ghe90, CG94, CMMS94]).

Notice that $(\forall \mathsf{E} =)$ allows one to equate two terms $f'\{A'\}$ and $f''\{A''\}$ even when A' and A'' are not the same type, and it expresses a sort of "irrelevance" of the argument type in second order application. This form of the rule was first defined in [CMMS94], where the interested reader can find a discussion on its motivations.

By exploiting the alternative definition of the semantics (see Remark 10.14) it is easy to see that it validates the proposed equational system. First we need a simple technical lemma which is the semantical counterpart of weakening.

Lemma 10.15 *Let* $\Gamma, \Delta, \Delta' \vdash b : B$ *and let* $\Gamma, \Delta, x{:}A, \Delta' \vdash \diamondsuit$. *Then for any* $\langle \gamma, \delta \rangle \in [\![\Gamma, \Delta, \Delta']\!]$ *and* $v \in [\![A]\!]\gamma$

$$[\![\Gamma, \Delta, \Delta' \vdash b : B]\!]\langle \gamma, \delta \rangle = [\![\Gamma, \Delta, x{:}A, \Delta' \vdash b : B]\!]\langle \gamma, \delta[x \mapsto v] \rangle$$

(Notice that $\Gamma, \Delta, x{:}A, \Delta' \vdash b : B$ *is derivable by Lemma 4.11.)*

Proof. Trivial induction on the structure of b. □

Theorem 10.16 (soundness of deduction) *If the judgement* $\Gamma, \Delta \vdash a = b : A$ *is derivable in the equational system of* F-bounded *then, for any* $\langle \gamma, \delta \rangle \in [\![\Gamma, \Delta]\!]$ *we have* $[\![\Gamma, \Delta \vdash a : A]\!]\langle \gamma, \delta \rangle = [\![\Gamma, \Delta \vdash b : A]\!]\langle \gamma, \delta \rangle$.

Proof. The proof can be done by straightforward induction on the structure of the derivation d of $\Gamma, \Delta \vdash a = b : A$ and by cases on the last rule applied in d. The cases of $(\mathsf{Refl} =)$, $(\mathsf{Symm} =)$ and $(\mathsf{Trans} =)$ and of structural rules are trivial. The case of rule $(\mathsf{Top} =)$ is an immediate consequence of Corollary 10.11, since $[\![\mathit{Top}]\!]\gamma$ has only one equivalence class. The only interesting cases are rule $(\forall \mathsf{E} =)$ and rules β and η for terms and types.

- $(\forall \mathsf{E} =)$ We must prove that

  $$[\![\Gamma, \Delta \vdash f'\{A'\} : C]\!]\langle \gamma, \delta \rangle = [\![\Gamma, \Delta \vdash f''\{A''\} : C]\!]\langle \gamma, \delta \rangle.$$

  By induction hypothesis

  $$[\![\Gamma, \Delta \vdash f' : \forall \alpha \leq A.\, B]\!]\langle \gamma, \delta \rangle = v = [\![\Gamma, \Delta \vdash f'' : \forall \alpha \leq A.\, B]\!]\langle \gamma, \delta \rangle.$$

$$\frac{\Gamma,\Delta \vdash \lambda x{:}A.\, b : A \to B \quad \Gamma,\Delta \vdash a : A}{\Gamma,\Delta \vdash (\lambda x{:}A.\, b)(a) = b[x \leftarrow a] : B}\ (\beta\mathsf{Term} =)$$

$$\frac{\Gamma,\Delta \vdash b : A \to B}{\Gamma,\Delta \vdash \lambda x{:}A.\, b(x) = b : A \to B}\ (\eta\mathsf{Term} =)$$

$$\frac{\Gamma,\Delta \vdash \Lambda\alpha \leq A.\, b : \forall\alpha \leq A.\, B \quad \Gamma \vdash A' \leq A[\alpha \leftarrow A']}{\Gamma,\Delta \vdash (\Lambda\alpha \leq A.\, b)A' = b[\alpha \leftarrow A'] : B[\alpha \leftarrow A']}\ (\beta\mathsf{Type} =)$$

$$\frac{\Gamma,\Delta \vdash b : \forall\alpha \leq A.\, B}{\Gamma,\Delta \vdash \Lambda\alpha \leq A.\, b\{\alpha\} = b : \forall\alpha \leq A.\, B}\ (\eta\mathsf{Type} =)$$

$$\frac{\Gamma,\Delta \vdash a : A}{\Gamma,\Delta \vdash a = a : A}\ (\mathsf{Refl} =) \qquad \frac{\Gamma,\Delta \vdash a = b : A}{\Gamma,\Delta \vdash b = a : A}\ (\mathsf{Symm} =)$$

$$\frac{\Gamma,\Delta \vdash a = b : A \quad \Gamma,\Delta \vdash b = c : A}{\Gamma,\Delta \vdash a = c : A}\ (\mathsf{Trans} =)$$

$$\frac{\Gamma,\Delta, x{:}A \vdash a = b : B}{\Gamma,\Delta \vdash \lambda x{:}A.\, a = \lambda x{:}A.\, b : A \to B}\ (\to \mathsf{I} =)$$

$$\frac{\Gamma,\Delta \vdash f' = f'' : A \to B \quad \Gamma,\Delta \vdash a' = a'' : A}{\Gamma,\Delta \vdash f'(a') = f''(a'') : B}\ (\to \mathsf{E} =)$$

$$\frac{\Gamma,\alpha \leq A,\Delta \vdash a = b : B}{\Gamma,\Delta \vdash \Lambda\alpha \leq A.\, a = \Lambda\alpha \leq A.\, b : \forall\alpha \leq A.\, B}\ (\forall \mathsf{I} =)$$

$$\frac{\begin{array}{c}\Gamma,\Delta \vdash f' = f'' : \forall\alpha \leq A.\, B \qquad \Gamma \vdash A' \leq A[\alpha \leftarrow A'] \\ \Gamma \vdash A'' \leq A[\alpha \leftarrow A''] \qquad \Gamma \vdash B[\alpha \leftarrow A'], B[\alpha \leftarrow A''] \leq C\end{array}}{\Gamma,\Delta \vdash f'\{A'\} = f''\{A''\} : C}\ (\forall \mathsf{E} =)$$

$$\frac{\Gamma,\Delta \vdash a : \mathit{Top} \quad \Gamma,\Delta \vdash b : \mathit{Top}}{\Gamma,\Delta \vdash a = b : \mathit{Top}}\ (\mathsf{Top} =)$$

Table 2: The equational system.

By Remark 10.14, we have $[\![\Gamma, \Delta \vdash f'\{A'\} : B[\alpha{\leftarrow}A']]\!]\langle\gamma, \delta\rangle = [i]_{[\![B[\alpha\leftarrow A']]\!]\gamma}$, where i is any index in v. Since $\Gamma \vdash B[\alpha{\leftarrow}A'] \leq C$, again by the same corollary we conclude

$$[\![\Gamma, \Delta \vdash f'\{A'\} : C]\!]\langle\gamma, \delta\rangle = [i]_{[\![C]\!]\gamma}.$$

By an analogous reasoning $[\![\Gamma, \Delta \vdash f''\{A''\} : C]\!]\langle\gamma, \delta\rangle = [i]_{[\![C]\!]\gamma}$ and thus we can conclude.

- ($\beta\mathsf{Term}=$) For any pair of indexes $i \in [\![\Gamma, \Delta \vdash \lambda x{:}A.\, b : A{\to}B]\!]\langle\gamma, \delta\rangle$ and $j \in v = [\![\Gamma, \Delta \vdash a : A]\!]\langle\gamma, \delta\rangle$ we have that:

$$\begin{aligned}
&[\![\Gamma, \Delta \vdash (\lambda x{:}A.\, b)(a) : B]\!]\langle\gamma, \delta\rangle = \\
&\quad = [i.j]_{[\![B]\!]\gamma} && \text{[by term interpretation, case } (\to \mathsf{E})] \\
&\quad = [\![\Gamma, \Delta, x{:}A \vdash b : B]\!]\langle\gamma, \delta[x{\mapsto}v]\rangle && \text{[by } i \in [\![\Gamma, \Delta \vdash \lambda x{:}A.\, b : A{\to}B]\!]\langle\gamma, \delta\rangle \\
&&& \text{and term interpretation, case } (\to \mathsf{I})] \\
&\quad = [\![\Gamma, \Delta \vdash b[x{\leftarrow}a] : B]\!]\langle\gamma, \delta\rangle && \text{[by Lemma 10.9(2)]}
\end{aligned}$$

We conclude by observing that the quantification over i, j is not trivial thanks to Corollary 10.12.

- ($\eta\mathsf{Term}=$) We must prove that:

$$[\![\Gamma, \Delta \vdash \lambda x{:}A.\, b(x) : A{\to}B]\!]\langle\gamma, \delta\rangle = [\![\Gamma, \Delta \vdash b : A{\to}B]\!]\langle\gamma, \delta\rangle$$

First of all notice that, by definition, $[\![\Gamma, \Delta \vdash \lambda x{:}A.\, b(x) : A{\to}B]\!]\langle\gamma, \delta\rangle = [i]_{[\![A\to B]\!]\gamma}$ for any $i \in \omega$ such that

$$\forall v \in \mathbf{Q}([\![A]\!]\gamma),\ \forall j \in v,\ i.j \in [\![\Gamma, \Delta, x{:}A \vdash b(x) : B]\!]\langle\gamma, \delta[x{\mapsto}v]\rangle. \quad (\ddagger)$$

Now, taking any $i \in [\![\Gamma, \Delta \vdash b : A{\to}B]\!]\langle\gamma, \delta\rangle$, to conclude we just have to prove that it satisfies condition ($\ddagger$). Since x is not free in b, its semantics does not change if we update the value of x in δ. Formally, by Lemma 10.15, $\forall v \in \mathbf{Q}([\![A]\!]\gamma)$, $i \in [\![\Gamma, \Delta, x : A \vdash b : A{\to}B]\!]\langle\gamma, \delta[x{\mapsto}v]\rangle$. For any $j \in v$, by the semantics of variables, we have that $j \in [\![\Gamma, \Delta, x : A \vdash x : A]\!]\langle\gamma, \delta[x{\mapsto}v]\rangle$ and hence $i.j \in [\![\Gamma, \Delta, x{:}A \vdash b(x) : B]\!]\langle\gamma, \delta[x{\mapsto}v]\rangle$, by $(\to \mathsf{E})$.

- ($\beta\mathsf{Type}=$), ($\eta\mathsf{Type}=$) In this case the correctness immediately follows from the observation that, by Lemma 10.10, the semantics of terms just depends on the erasure and on the type of the term. Then simply observe that such rules equate terms with the same erasure. □

The previous theorem has soundness of reduction as an immediate corollary, namely, if a is a closed *F-bounded* term, such that $\vdash a : A$ and $a \twoheadrightarrow b$, then a and b have the same semantics (as elements of type A). In fact, it is sufficient to observe that in this case $\vdash a = b : A$ and then apply Theorem 10.16.

Finally, we observe that the semantics defined is consistent. To this aim we use the type $\mathit{Bool} = \forall\alpha \leq \mathit{Top}.\, \alpha{\to}\alpha{\to}\alpha$, which is the usual encoding of Church's booleans in system F, and we consider the two closed normal form terms of type *Bool*:

$$\mathit{true} \equiv \Lambda\alpha \leq \mathit{Top}.\, \lambda x{:}\alpha.\, \lambda y{:}\alpha.\, x \qquad \mathit{false} \equiv \Lambda\alpha \leq \mathit{Top}.\, \lambda x{:}\alpha.\, \lambda y{:}\alpha.\, y$$

It is easy to see that $[\![\vdash \mathit{true} : \mathit{Bool}]\!] \neq [\![\vdash \mathit{false} : \mathit{Bool}]\!]$. In fact by definition, $i[\![\mathit{Bool}]\!]j \Leftrightarrow \forall p \in \mathit{PER}.\ \forall k,l,m,n \in \omega.\ k\ p\ l\ \wedge\ m\ p\ n \Rightarrow i.k.m\ p\ j.l.n$. Recall that $[\![\vdash \mathit{true} : \mathit{Bool}]\!]$ and $[\![\vdash \mathit{false} : \mathit{Bool}]\!]$ are the equivalence classes in Bool of the indexes of the binary projections on the first and on the second component respectively. To conclude it suffices to consider the p.e.r. $p = \{\langle 0,0\rangle, \langle 1,1\rangle\}$ and let $k = l = 0$, $m = n = 1$ (in the same way we may also prove that $\mathbf{Q}([\![\mathit{Bool}]\!])$ only contains $[\![\vdash \mathit{true}]\!]$ and $[\![\vdash \mathit{false}]\!]$).

11 Conclusions

In this paper we have studied some aspects of the theory of system *F-bounded*, concerning type and subtype checking, its relationship with system $F_{\leq}$, and its semantics. We have proved the following results:

- transitivity elimination, hence correctness and completeness of the standard subtype checking semi-algorithm;
- correctness and completeness of the standard type checking semi-algorithm;
- subject reduction for $\beta\eta$ reduction;
- characterization of type equivalence as the equivalence obtained by identifying $\alpha \leq \alpha$ with $\alpha \leq \mathit{Top}$ bounds;
- characterization of the relationship between system *F-bounded* and its variants *F-bounded*$^-$ and *F-bounded*$_{\leq}$;
- conservativity of *F-bounded* subtyping with respect to $F_{\leq}$, which implies that subtype checking, hence type checking, for system *F-bounded* is undecidable, and that an extension of system *F-bounded* with strong recursive types is non-conservative;
- coherence and consistency of a p.e.r. interpretation of system *F-bounded*, soundness of the term formation, subtyping, typing, reduction and equivalence rules with respect to this interpretation.

Termination of $\beta\eta$ reduction has not been investigated, since the result is already known from [Ghe97].

Although system *F-bounded* is more powerful than system $F_{\leq}$, essentially the same techniques can be used to prove analogous properties in the two systems. Some minor differences are due to the different shape of the $(\forall\ \leq)$ rule, but the conservativity result of Section 9 shows that an *F-bounded*-like version of that rule could have been adopted for system $F_{\leq}$ as well. This fact suggests the idea of viewing both systems as special cases of a wider family based on a conditional quantification $\forall\alpha/P(\alpha).\,T$ with corresponding introduction, elimination and subtyping rules, such as:

$$\frac{\Gamma, \Delta \vdash f : \forall\alpha/P(\alpha).\,B \quad \Gamma \vdash P(A')}{\Gamma, \Delta \vdash f\{A'\} : B[\alpha{\leftarrow}A']}\ (\forall_{\mathsf{p}}\mathsf{E})$$

$$\frac{\Gamma, P'(\alpha) \vdash P(\alpha) \quad \Gamma, P'(\alpha) \vdash B \leq B'}{\Gamma \vdash (\forall\alpha/P(\alpha).\,B) \leq (\forall\alpha/P'(\alpha).\,B')}\ (\forall_{\mathsf{p}}\leq)$$

Therefore, it may be interesting to investigate the possibility of defining some general language for predicates $P(\alpha)$ ensuring that the crucial properties of system $F_{\leq}$ are preserved.

In our opinion an interesting open issue is the study of the subtype checking of a kernel-fun variant of system *F-bounded*, i.e., a system where universal types are compared through the following weak rule.

$$\frac{\Gamma, \alpha \leq A' \vdash A \sim A' \quad \Gamma, \alpha \leq A' \vdash B \leq B'}{\Gamma \vdash (\forall \alpha \leq A.\, B) \leq (\forall \alpha \leq A'.\, B')} \ (\mathsf{kf}\ \forall \leq)$$

The kernel-fun variant of system $F_{\leq}$ is known to be decidable. We conjecture that the analogous variant of system *F-bounded* would be decidable too.

The kernel-fun variant of system *F-bounded* is interesting because its subtype theory should be simpler to deal with, and its expressive power not far from the power of the full system. In practice, the two systems differ above all in the treatment of existential bounded quantifiers. Existential quantifiers can be encoded in terms of universal ones, and the resulting subtyping rule turns out to be invariant in the bounds for the kernel-fun version, and covariant for the full version [GP98]. While the kernel-fun version of the universal quantification is powerful enough for practical aims, the kernel-fun version of existential quantification turns out to be weak in some specific situations. A typical example is given by the four different interpretations of object-oriented languages discussed in [BCP99], where the kernel-fun subtyping rule for existential types is shown to be expressive enough for the first three encodings, but too weak for the most expressive "ORBE" interpretation.

Another decidable variant of system $F_{\leq}$ is the one without a *Top* type [Kat92]. Hence a natural question regards the decidability of a variant of system *F-bounded* without the *Top* type and with no $\alpha \leq \alpha$ bound. However, this is a much less interesting question, since the system without *Top* is not as natural and expressive as the kernel-fun variation. The essential problem is that records with width subtyping cannot be encoded in this variant of the system, and, if they are added as primitive constructions, then decidability is lost.

To conclude we remark that, while here we have studied the pure system *F-bounded*, with no notion of value or type level recursion, a practical object-oriented language should contain both of them. Especially interesting is the study of type level recursion.

Strong and weak type level recursion, as defined in Section 9, have different peculiarities and raise different problems. In any case, both of them destroy the normalization property of β reduction, since they allow untyped lambda calculus terms to be easily encoded as terms of type $\mu X.X \to X$.

Strong recursion interferes with transitivity elimination [Ghe93], and thus with the completeness of the standard type checking algorithm, even for terms where no recursive type is used. The definition of complete type and subtype checking algorithms for second order systems with subtyping and strong recursion is still an open problem. The only known result is the algorithm for system kernel-fun defined in [CG99]. On the other hand, weak recursion does not modify the subtype relation, and has no effect on type checking since the type of a $\mathit{fold}_{\mu X.A}$ or $\mathit{unfold}_{\mu X.A}$ function can be read from its index, thus allowing these functions to be type-checked like any user-defined function. However, weak recursion is not a good match for *F-bounded* quantification. For instance, the type *Point* discussed in Section 2, if

defined via weak recursion, does not satisfy the condition:

$$\alpha \leq [x : Int; eq : \alpha \to Bool],$$

since a weak recursive type is a subtype only of other recursive types. This observation suggests that it may be interesting to explore some intermediate kind of recursion. For example, a notion of recursive types could be investigated, which is based on implicit unfolding ($\mu X.A \leq A[X \leftarrow \mu X.A]$) and explicit folding through a function $fold_{\mu X.A}$: $A[X \leftarrow \mu X.A] \to \mu X.A$.

From a semantic point of view, adding any kind of recursion would require the definition of a different interpretation. The realizability interpretation we presented would still be the basis of the semantics, but the domain of p.e.r.'s would have to be enriched with enough structure in order to deal with partiality and fix point definitions [Ama88, Car89, Ama91, AP90].

Acknowledgements

We are grateful to the anonymous referees for insightful and constructive remarks. This work has been partially supported by Esprit Working Groups 26142 - Applied Semantics and 22552 - PASTEL, and by Italian MURST, project InterData.

Appendix: The De Bruijn notation

In the paper we essentially adopt the De Bruijn approach for the treatment of variables. The idea consists in representing each variable occurrence as a pointer to the λ (or Λ) which binds the variable, hereafter referred to as the *binder* of the variable.

Concretely, in a term, an occurrence of a variable is represented as an integer index expressing the number of lambdas between the occurrence and the binder for the variable. More precisely, the index counts the number of lambdas whose scope includes the variable occurrence and which are in the scope of the binder. This leads to the so-called *nameless* term. Here is an untyped term and the corresponding nameless term.

$$\lambda x.\, \lambda y.\, x(\lambda z.\, xz)y \qquad\qquad \lambda.\, \lambda.\, 1(\lambda.\, 2\, 0)0.$$

The same technique can be extended to deal with our typed terms, possibly inside an environment. Bindings of the environment are treated exactly like λ or Λ bindings. Without going into further details we show some examples. For the reader's convenience we consider different indexes for value and type variables (denoted by n_v and n_t, respectively). The index represents, for value variables, the number of λ's and, for type variables, the number of Λ's (or $\forall$'s), between the variable occurrence and the binder of the variable. For instance $\Lambda\alpha \leq \mathit{Top}.\, \lambda x{:}\alpha \to \alpha.\, \lambda y{:}\alpha.\, xy$ becomes $\Lambda \leq \mathit{Top}.\, \lambda{:}0_t \to 0_t.\, \lambda{:}0_t.\, 1_v 0_v$, and $\alpha \leq \mathit{Top}, \beta \leq \alpha \to \beta,\ x:\alpha, y:\beta \vdash yx$ becomes $\leq \mathit{Top}, \leq 1_t \to 0_t,\ :1_t, :0_t \vdash 0_v 1_v$.

As highlighted in Section 3, working directly on De Bruijn indexes may be notationally too inconvenient. Therefore we continue using variable names, implicitly assuming that they are just a more convenient way of denoting De Bruijn indexes. In this way there is obviously a gap between what is written and what *should* be written by explicitly using the De Bruijn notation. To convince the reader that this gap can be easily filled in, let us present some of the basic definitions in the De Bruijn notation.

First of all a *free variable* in a nameless term is a pointer to a non-existing binder. More precisely an index n, if greater than the number k of nested binders having the index in their scope, represents the $n - k^{th}$ free variable. We can represent free variables in a term by using such numbers and write:

$$FV(n) = \{n\} \qquad FV(A \to B) = FV(A) \cup FV(B)$$
$$FV(\mathit{Top}) = \emptyset \qquad FV(\forall \alpha \leq A.\, B) = \{n - 1 \mid n \in FV(A) \cup FV(B) \ \wedge\ n > 0\}$$

Given an environment $\Gamma \equiv \leq A_1, \ldots, \leq A_n$, instead of collecting the set of the variables defined in Γ, we simply count the number of such variables, i.e., we define:

$$\mathit{vars}(\Gamma) = |\Gamma| = n.$$

The rules for well-formedness of type environments become:

$$\epsilon \vdash \diamondsuit \quad (\epsilon\mathsf{TEnv}) \qquad\qquad \frac{\Gamma \vdash \diamondsuit \quad \max(FV(A)) \leq \mathit{vars}(\Gamma) + 1}{\Gamma, \leq A \vdash \diamondsuit}\ (\mathsf{TEnv})$$

The other rules have to be changed in a similar way.

References

[AC93] Roberto M. Amadio and Luca Cardelli. Subtyping recursive types. *ACM Transactions on Programming Languages and Systems*, 15(4):575–631, 1993. A preliminary version appeared in POPL '91 (pp. 104–118) and as DEC Systems Research Center Research Report number 62, August 1990.

[AC96a] M. Abadi and L. Cardelli. On subtyping and matching. *ACM Transactions on Programming Languages and Systems*, 18(4):401–423, 1996.

[AC96b] M. Abadi and L. Cardelli. *A Theory of Objects*. Springer-Verlag, 1996.

[Ama88] R. Amadio. A fixed point extension of the second order lambda calculus: observational equivalences and models. In *Proc. IEEE Logic in Comp. Sci.*, pages 51–60, 1988.

[Ama91] Roberto M. Amadio. Recursion over realizability structures. *Information and Computation*, 91(1):55–85, March 1991.

[AP90] M. Abadi and G. D. Plotkin. A per model of polymorphism and recursive types. In *Proceedings, Fifth Annual IEEE Symposium on Logic in Computer Science*, pages 355–365, Philadelphia, Pennsylvania, 4–7 June 1990. IEEE Computer Society Press.

[BCC+96] K. B. Bruce, L. Cardelli, G. Castagna, the Hopkins Objects Group (J. Eifrig, S. Smith, V. Trifonov), G. T. Leavens, and B. Pierce. On binary methods. *Theory and Practice of Object Systems*, 1(3):221–242, 1996.

[BCP99] Kim B. Bruce, Luca Cardelli, and Benjamin C. Pierce. Comparing object encodings. *Information and Computation*, 1999. To appear in a special issue with papers from *Theoretical Aspects of Computer Software (TACS)*, September, 1997.

[BL90] K. B. Bruce and G. Longo. A modest model of records, inheritance, and bounded quantification. *Information and Computation*, 87:196–239, 1990.

[BMM90] K. B. Bruce, A. R. Meyer, and J. C. Mitchell. The semantics of second-order lambda calculus. In Gérard Huet, editor, *Logical Foundations of Functional Programming*, University of Texas at Austin Year of Programming Series, pages 213–272. Addison-Wesley, 1990. Also appeared in *Information and Computation* 84, 1 (January 1990).

[Bru94] K. B. Bruce. A paradigmatic object-oriented programming language: Design, static typing and semantics. *Journal of Functional Programming*, 4(2), April 1994. A preliminary version appeared in POPL 1993 under the title "Safe Type Checking in a Statically Typed Object-Oriented Programming Language".

[BSvG95] K. B. Bruce, A. Schuett, and R. van Gent. PolyTOIL: A type-safe polymorphic object-oriented language. In W. Olthoff, editor, *Proceedings of* ECOOP '95, LNCS 952, pages 27–51, Aarhus, Denmark, August 1995. Springer-Verlag.

[Car89] Felice Cardone. Relational semantics for recursive types and bounded quantification. In *Proceedings of the Sixteenth International Colloquium on Automata, Languages, and Programming*, volume 372 of *Lecture Notes in Computer Science*, pages 164–178, Stresa, Italy, July 1989. Springer-Verlag.

[Car90] L. Cardelli. Notes about $F^{\omega}_{<:}$. Unpublished manuscript, October 1990.

[Cas96] G. Castagna. Integration of parametric and "ad hoc" second order polymorphism in a calculus with subtyping. *Formal Aspects of Computing*, 8(3):247–8293, 1996.

[Cas97] G. Castagna. Unifying overloading and λ-abstraction: $\lambda^{\{\}}$. *Theoretical Computer Science*, 176(1–2):337–345, 1997.

[CCH+89] P. Canning, W. Cook, W. Hill, W. Olthoff, and J. Mitchell. F-bounded quantification for object-oriented programming. In *Fourth International Conference on Functional Programming Languages and Computer Architecture*, pages 273–280, September 1989.

[CG92] P.-L. Curien and G. Ghelli. Coherence of subsumption: Minimum typing and type-checking in $F_{\leq}$. *Mathematical Structures in Computer Science*, 2:55–91, 1992. Also in C. A. Gunter and J. C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994).

[CG94] P.-L. Curien and G. Ghelli. Decidability and confluence of $\beta\eta\mathrm{top}_{\leq}$ reduction in $F_{\leq}$. *Information and Computation*, 109(1, 2):57–114, 1994.

[CG99] D. Colazzo and G. Ghelli. Subtyping recursive types in kernel Fun, extended abstract. In *Proc. of the 14th Annual IEEE Symposium on Logic in Computer Science (LICS), Trento, Italy*, June 1999.

[CGL93] G. Castagna, G. Ghelli, and G. Longo. A semantics for $\lambda\&$-early: a calculus with overloading and early binding. In M. Bezen and J.F. Groote, editors, *Proc. of the International Conference on Typed Lambda Calculi and Applications (TLCA), Utrecht, The Netherlands*, number 664 in Lecture Notes in Computer Science, pages 107–123, Berlin, March 1993. Springer-Verlag.

[CGL95] G. Castagna, G. Ghelli, and G. Longo. A calculus for overloaded functions with subtyping. *Information and Computation*, 117(1):115–135, 1995.

[CL91] L. Cardelli and G. Longo. A semantic basis for Quest. *Journal of Functional Programming*, 1(4):417–458, October 1991. Preliminary version in ACM Conference on Lisp and Functional Programming, June 1990. Also available as DEC SRC Research Report 55, Feb. 1990.

[CMMS94] L. Cardelli, S. Martini, J. C. Mitchell, and A. Scedrov. An extension of system F with subtyping. *Information and Computation*, 109(1–2):4–56, 1994. A preliminary version appeared in TACS '91 (Sendai, Japan, pp. 750–770).

[Coo91] W. Cook. Object-oriented programming versus abstract data types. In J. W. de Bakker et al., editors, *Foundations of Object-Oriented Languages*, volume 489 of *Lecture Notes in Computer Science*, pages 151–178. Springer-Verlag, 1991.

[CW85] L. Cardelli and P. Wegner. On understanding types, data abstraction, and polymorphism. *ACM Computing Surveys*, 17(4):471–522, December 1985.

[dB72] N. G. de Bruijn. Lambda-calculus notation with nameless dummies: a tool for automatic formula manipulation with application to the Church-Rosser theorem. *Indag. Math.*, 34(5):381–392, 1972.

[Ghe90] G. Ghelli. *Proof Theoretic Studies about a Minimal Type System Integrating Inclusion and Parametric Polymorphism.* PhD thesis, Università di Pisa, March 1990. Technical report TD–6/90, Dipartimento di Informatica, Università di Pisa.

[Ghe91] G. Ghelli. A static type system for late binding overloading. In A. Paepcke, editor, *Proc. of the Sixth Intl. ACM Conference on Object Oriented Programing: Systems, Languages, and Applications (OOPSLA), Phoenix, Arizona*, number 26 (11) in ACM SIGPLAN Notices, pages 129–145, Reading, MA, 1991. Addison-Wesley.

[Ghe93] G. Ghelli. Recursive types are not conservative over $F_{\leq}$. In M. Bezen and J.F. Groote, editors, *Proc. of the International Conference on Typed Lambda Calculi and Applications (TLCA), Utrecht, The Netherlands*, number 664 in Lecture Notes in Computer Science, pages 146–162, Berlin, March 1993. Springer-Verlag.

[Ghe95] G. Ghelli. Divergence of $F_{\leq}$ type checking. *Theoretical Computer Science*, 139(1-2):131–162, 1995.

[Ghe97] G. Ghelli. Termination of system F-bounded: A complete proof. *Information and Computation*, 139(1):39–56, 1997.

[Gir72] J.-Y. Girard. *Interprétation fonctionelle et élimination des coupures dans l'arithmétique d'ordre supérieur.* PhD thesis, University of Paris VII, 1972.

[GMW79] M.H. Gordon, R.M. Milner, and C. Wadsworth. *Edinburgh LCF*, volume 78 of *Lecture Notes in Computer Science*. Springer-Verlag, 1979.

[GP98] G. Ghelli and B. Pierce. Bounded existentials and minimal typing. *Theoretical Computer Science*, 193(1-2):75–96, 1998.

[Gun92] C. A. Gunter. *Semantics of Programming Languages: Structures and Techniques.* Foundations of Computing. The MIT Press, 1992.

[HP96] M. Hofmann and B. C. Pierce. Positive subtyping. *Information and Computation*, 126(1):11–33, 1996.

[Kat92] D. Katiyar. Subtyping F-bounded types. In *ANSA Workshop on F-bounded quantification, Cambridge*, 1992.

[KS92] D. Katiyar and S. Sankar. Completely bounded quantification is decidable. In *Proceedings of the ACM SIGPLAN Workshop on ML and its Applications*, June 1992.

[MKO95] D. McAllester, J. Kučan, and D. F. Otth. A proof of strong normalization of F_2, F_ω and beyond. *Information and Computation*, 121(2):193–200, 1995.

[MP88] J. Mitchell and G. Plotkin. Abstract types have existential type. *ACM Transactions on Programming Languages and Systems*, 10(3):470–502, July 1988.

[Pie94] B. C. Pierce. Bounded quantification is undecidable. *Information and Computation*, 112(1):131–165, July 1994. Also in C. A. Gunter and J. C. Mitchell, editors, *Theoretical Aspects of Object-Oriented Programming: Types, Semantics, and Language Design* (MIT Press, 1994). A preliminary version appeared in POPL '92.

[Pie97] Benjamin C. Pierce. Intersection types and bounded polymorphism. *Mathematical Structures in Computer Science*, 7(2):129–193, April 1997.

[PS97] Benjamin Pierce and Martin Steffen. Higher-order subtyping. *Theoretical Computer Science*, 176(1–2):235–282, 20 April 1997.

[Rey74] J. Reynolds. Towards a theory of type structure. In *Proc. Colloque sur la Programmation*, pages 408–425, New York, 1974. Springer-Verlag LNCS 19.