Subtyping Dependent Types

David Aspinall
<da@dcs.ed.ac.uk>
LFCS, Division of Informatics
University of Edinburgh
King's Buildings
Edinburgh EH9 3JZ, U. K.

Adriana Compagnoni*
<abc@cs.stevens-tech.edu>
Department of Computer Science
Stevens Institute of Technology
Castle Point on Hudson
Hoboken, NJ 07030, U. S. A.

February 8, 2000

Abstract

The need for subtyping in type-systems with dependent types has been realized for some years. But it is hard to prove that systems combining the two features have fundamental properties such as subject reduction. Here we investigate a subtyping extension of the system λP, which is an abstract version of the type system of the Edinburgh Logical Framework LF. By using an equivalent formulation, we establish some important properties of the new system $\lambda P_{\leq}$, including subject reduction. Our analysis culminates in a complete and terminating algorithm which establishes the decidability of type-checking.

This is an expanded version of the paper which appeared under the same title in *Proc. 11th Annual Symposium on Logic in Computer Science*, IEEE 1996.

Keywords: **type theory, dependent types, subtyping**

Contents

1 Introduction **2**
1.1 Summary of application areas . . . 3
1.2 Combining subtyping and dependent types 4

2 The system $\lambda P_{\leq}$ **6**
2.1 Basic properties of $\lambda P_{\leq}$ 9
2.2 Towards subject reduction 14

3 A subtyping algorithm **15**
3.1 Closure under β_2-reduction 17
3.2 Soundness 18
3.3 Reflexivity and transitivity 19
3.4 Completeness 23
3.5 Equivalence 24
3.6 Decidability 25

4 A type-checking algorithm **26**
4.1 Basic properties of the algorithm . 27
4.2 Equivalence 28
4.3 Decidability 32

5 Conclusion **34**

*Most of this work was done while Adriana Compagnoni was at the LFCS, Division of Informatics, University of Edinburgh, King's Buildings, Edinburgh EH9 3JZ, U. K., and at the University of Cambridge, Computer Laboratory, New Museums Site, Pembroke Street, Cambridge CB2 3QG, U. K.

1 Introduction

Subtyping captures concepts from diverse areas of computer science. If A and B are sets, then $A \leq B$ ('A is a subtype of B') means that elements of A are also elements of B. If A and B are specifications, then programs satisfying the specification A also satisfy B. In object-oriented programming, if A and B are object descriptions, then $A \leq B$ states that where an object with interface B is expected, it is safe to use an object with interface A. If A and B are theorems, then a proof of A is also a proof of B. Understanding the essence, subtleties, and general properties of subtyping illuminates a wide area.

Dependent types are types which depend on terms. A typical example is $\mathsf{List}(n)$, the type of lists of length n. Dependent types are more expressive than simple types: the functional map can be given the type $\pi n{:}\mathsf{Nat}.\, \mathsf{List}(n) \to \mathsf{List}(n)$, expressing that it is parametric in the length of lists it is applied to. More generally, type dependency can express a relationship between the input of a function and its output, which can be used to specify its behaviour. Dependent types also facilitate the encoding of logics via the *judgements-as-types* paradigm of the Edinburgh Logical Framework LF [17]. Suppose p is a term which encodes a formula of some logic. Then the dependent type $\mathsf{True}(p)$ corresponds to a truth judgement and its elements encode proofs of p. The encoded proofs are constructed from constants that encode the axioms and rules of the logic.

There are several application areas where researchers have discovered a need to combine subtyping and dependent types. In the next section we shall give an overview of these applications; here we sketch a typical example of logic representation. (We assume some familiarity with LF; another example describing datatypes for a programming language is mentioned in Section 2.)

The example is a formal system for the call-by-value λ-calculus, taken from [4]. The syntax of the call-by-value λ-calculus is the same as that of the traditional λ-calculus, but it has a restricted rule of β-equality:

$$(\lambda x.\, M)\ N = M[x := N] \qquad \text{provided } N \text{ is a value}$$

where a *value* is a variable or an abstraction. The restriction is achieved in LF by massaging the syntax of the encoded λ-terms. Two syntactic categories are declared:

$$\begin{array}{ll} o & : \star \\ v & : \star \end{array}$$

(these are types in LF; $\star$ is the kind of types).

The intention is that o is the type of all expressions whilst v is a subset of o corresponding to the expressions which are values. The λ-constructor, lda, binds terms of type v and such terms can only be variables or other terms constructed with lda. An extra constructor "!" is needed, which can be thought of as an injection function from values to expressions:

$$\begin{array}{ll} ! & : v \to o \\ lda & : (v \to o) \to v \\ app & : o \to o \to o \end{array}$$

For the proof system, there is an equality judgement together with constants representing axioms and rules:

$$\begin{array}{ll} = & : o \to o \to \star \\ E_{refl} & : \prod_{x:o} x = x \\ \vdots & \\ E_\beta & : \prod_{m:v\to o, n:v} app\ (!(lda\ m))\ (!n) = mn \end{array}$$

But the injection function "!" is a big nuisance. It pervades the encoding of terms yet it corresponds to nothing in the original syntax. Lambda expressions become more difficult to read and write; the example mechanisation in *LEGO* given in [4] is testimony to this. Clearly when we use the encoding we would rather not mention the injection at all.

With subtyping, we simply declare v as a subtype of o:

$$\begin{array}{ll} o & : \star \\ v & \leq o : \star \end{array}$$

and then the injection function is not needed. In effect, it becomes *implicit*: we may imagine that it is inserted automatically wherever necessary. The β-rule now reads:

$$E_\beta \quad : \prod_{m:v\to o, n:v} app\ (lda\ m)\ n = mn$$

and we do not need any extra constructors.

1.1 Summary of application areas

Edinburgh Logical Framework The need for subtyping in a dependently typed lambda calculus was noticed during the Edinburgh LF project, around 1987. Mason pointed out that subtypes would be useful when representing Hoare's logic: one would like to treat the type of quantifier-free boolean expressions (used in programs) as a subtype of the type of first-order formulae (used in assertions), because formulae contain quantifiers that cannot appear in programs [20]. Without subtypes extra machinery is necessary, either encoding explicit coercion functions or additional judgements to express syntactic properties. Either device complicates the encoding. As we have demonstrated above with the call-by-value λ-calculus example, other common examples of encodings in LF also benefit from subtyping.

Later, Pfenning gave more cases of cumbersome encodings of syntax, and proposed a solution by extending LF with *refinement types*, a restricted form of subtypes [21]. Moreover, he demonstrated that refinement types (or subtyping) can allow a limited form of proof reuse, so that one proof term proves several judgements. (This is connected with the interpretation of subtyping as intuitionistic implication explained by Longo *et al.* [18].) Pfenning proved that his system is decidable and is a conservative extension of LF; see Section 5 for comparison with our work.

Other Applied Type Theories Pfenning's application was the proof assistant *Elf* which implements LF. A richer type theory is implemented by the *LEGO* system, in which researchers at Edinburgh and Erlangen recently tested Pierce and Turner's subtyping model

of object-orientation [24]. They extended the model to include proofs about objects and thus type-dependency. Because *LEGO* lacks subtyping, coercion functions are used, but it was found that inserting coercions quickly becomes tedious in practice.

Other applications in *LEGO* are easy to find. Subtyping is an important extension needed for proof assistants so that the formalization of mathematics can be brought closer to standard mathematical practice. Proof assistants such as *Elf*, *LEGO*, and their relatives *NuPrl*, *Coq* and *Alf* would all benefit from the addition of subtyping.

Type Systems for Programming Languages During the 1980's, Cardelli proposed several rich type systems for programming languages combining subtyping and type-dependency. The system in [9] is illustrated with examples of dependent datatypes and subtypings between them. At a workshop in 1986, Cardelli described ideas about type-checking techniques for these systems but at the outset accepted that the techniques would only lead to a semi-decision procedure, because of (for example) the combination of recursive types and type dependency [8].

We believe that our system is the first fragment of Cardelli's language, retaining subtyping and dependent types, to be shown to have a decidable type inference problem.

Type Systems for Specification Languages In algebraic specification, a language called ASL+ was proposed by Sannella, Sokołowski and Tarlecki [25] to model formal program development in-the-large. The types of ASL+ are algebraic specifications, terms are programs, and subtyping models specification refinement. Dependent types of the form $\pi x{:}A.\, B$ model specifications of parameterised programs (similar to functors in Standard ML); an implementation of $\pi x{:}A.\, B$ should map a program P satisfying A to a program satisfying $B[x := P]$.

The investigations of Sannella *et al.* into this language were preliminary and the progress reported here has fed into the continuation of their work in [2].

1.2 Combining subtyping and dependent types

In separation, subtyping and type-dependency have been well-studied. Yet their combination leads to systems that are difficult to study. We tread close to the line of undecidability, as in Cardelli's system or the second-order system $F_{\leq}$ [23]. Although it has been argued that semi-decision procedures may be acceptable in type-checkers for programming languages, decidability is essential for applied type theories where type-checking serves as proof-checking.

One thing that makes the study of these systems difficult is that with dependent types, the typing and subtyping relations become intimately tangled, which means that tested techniques of examining subtyping in isolation no longer apply.

Let us quickly show how typing and subtyping become tangled. The archetypal rule of subtyping is *subsumption*, which allows a term of a type A to be used where one of a supertype B is expected:

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A \leq B}{\Gamma \vdash M : B}$$

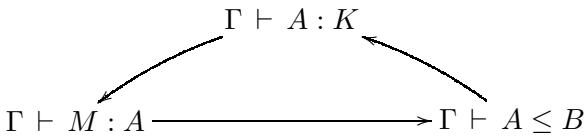
(as usual, Γ denotes a context of assumptions about the types of variables — see Section 2 below). So the typing judgement depends on the subtyping judgement. When a system has dependent types like List it must have a *kinding* rule to check that an application of a type-function to a term is well-formed:

$$\frac{\Gamma \vdash \mathsf{List} : \mathsf{Nat} \to \star \qquad \Gamma \vdash n : \mathsf{Nat}}{\Gamma \vdash \mathsf{List}(n) : \star}$$

So the kinding judgement depends on the typing judgement. We expect the subtyping relationship to hold *a priori* only between well-formed types; for example, inferring reflexivity of subtyping between types:

$$\frac{\Gamma \vdash A : \star}{\Gamma \vdash A \leq A}$$

So subtyping depends on typing, via kinding. As a picture:



Of course there is nothing bad about such a mutually recursive definition *per se*. But it turns out to significantly complicate our meta-theoretic study, compared with other well-understood subtyping systems (*e.g.,* [16, 22, 26, 13]) which lack this circularity.

In the remainder of this paper we study the addition of subtyping to the system λP, an abstract version of the type-system (sometimes called $\lambda\Pi$) which underlies LF [5, 17]. This is a pure system with type-valued functions dependent on terms. In Section 2 we define $\lambda\mathrm{P}_{\leq}$, showing examples of using the rules, and we prove some basic meta-theoretic properties.

At a certain point in the development of the meta-theory, things become difficult to analyse directly because of the circularity described above. So we design an algorithmic version of the subtyping relation which breaks the cycle of dependencies. The new relation does not depend on kinding, and only relates normal forms. But still there is a circularity, since we want to know that normalization steps used by the subtyping algorithm preserve kinding. To solve this we make another separation: β-reduction is split into two levels, β_1-reduction on terms and β_2-reduction on types. Type normalization only depends on β_2-reduction; at the outset we can prove rather more about this than about β_1-reduction. This untangles things enough to prove equivalence of the two subtyping relations, and then properties about the original relation. The analysis of subtyping is described in Section 3.

In Section 4 we describe the type-checking algorithm. We break more dependencies between the judgements and then we prove our main result: the algorithm is correct and terminates on all inputs, so $\lambda\mathrm{P}_{\leq}$ is decidable. A corollary is the minimal type property: every typable term possesses a minimal type in the subtype relation.

We believe that this work (first reported in [3]) describes the first proof of decidability for subtyping dependent types, in a system uniformly extended with a subsumption rule and a subtyping relation. In Section 5 we summarise the achievement and the related work, and mention some directions for further research.

2 The system $\lambda P_{\leq}$

The system $\lambda P_{\leq}$ (pronounced "lambda-pee-sub") is formally defined by the rules which follow below, also summarized at the end of the paper. The rules define four judgement forms:

$\Gamma \vdash K$	'K is a kind in context Γ'
$\Gamma \vdash A : K$	'type A has kind K in context Γ'
$\Gamma \vdash M : A$	'term M has type A in context Γ'
$\Gamma \vdash A \leq B$	'A is a subtype of B in context Γ'

In general, we use "type" to refer to both types (which have the kind $\star$) and type constructors (which have other kinds). We use the notation $\Gamma \vdash A, B : K$ to abbreviate the two judgements $\Gamma \vdash A : K$ and $\Gamma \vdash B : K$.

For those familiar with the description of λP in [5], we differ by using a stratified presentation separating the syntactic categories of kinds, types, and terms, and replacing the start and weakening rules by the kind formation judgement. This is close to the presentation of $\lambda\Pi$ in the appendix of [17].

The underlying grammar of *pre-terms* and *pre-contexts* is:

$$\begin{array}{rcl} M & ::= & x \mid \lambda x{:}A.\, M \mid MM \\ A & ::= & \alpha \mid \pi x{:}A.\, A \mid \Lambda x{:}A.\, A \mid AM \\ K & ::= & \star \mid \Pi x{:}A.K \\ \Gamma & ::= & \langle\rangle \mid \Gamma,\, x : A \mid \Gamma,\, \alpha : K \mid \Gamma,\, \alpha \leq A : K \end{array}$$

We assume throughout that pre-contexts never contain repeated declarations of the same variable.

Sometimes the letters $U, V, \ldots$ will be used to range over pre-terms which may be terms, types or kinds. Substitution is defined in the usual way for term variables $U[x := M]$ and type variables $U[\alpha := A]$. As mentioned, we distinguish two kinds of β-reduction:

$$\begin{array}{lll} \mathbf{C}[(\lambda x{:}A.\, M)N] & \longrightarrow_{\beta_1} & \mathbf{C}[M[x := N]] \\ \mathbf{C}[(\Lambda x{:}A.\, B)M] & \longrightarrow_{\beta_2} & \mathbf{C}[B[x := M]] \end{array}$$

($\mathbf{C}[-]$ indicates a pre-term with a hole in it). The union of the two reductions is written $\longrightarrow_{\beta}$. Generally, $\longrightarrow\!\!\!\!\rightarrow_R$ is the reflexive and transitive closure of the reduction $\longrightarrow_R$, and $=_R$ is the symmetric closure of $\longrightarrow\!\!\!\!\rightarrow_R$. The term U^R is the R-normal form of U.

Formation, kinding and typing are as in λP (or $\lambda\Pi$), except that the type conversion rule is replaced by subsumption, and we allow bounded type-variables in the context.

Here are the rules for kind and context **formation**:

$$\frac{}{\langle\rangle \vdash \star} \qquad \text{(F-EMPTY)}$$

$$\frac{\Gamma \vdash A : \star}{\Gamma,\, x : A \vdash \star} \qquad \text{(F-TERM)}$$

$$\frac{\Gamma \vdash K}{\Gamma, \alpha : K \vdash \star} \quad (\text{F-TYPE})$$

$$\frac{\Gamma \vdash A : K}{\Gamma, \alpha \leq A : K \vdash \star} \quad (\text{F-SUBTYPE})$$

$$\frac{\Gamma, x : A \vdash K}{\Gamma \vdash \Pi x{:}A.K} \quad (\text{F-}\Pi)$$

The kind of types is $\star$, which is always well-formed. The statement $\Gamma \vdash \star$ says that Γ is a well-formed context, avoiding the need for another judgement. The kind $\Pi x{:}A.K$ classifies type families, which map a term of type A to a type of kind K.

We have two ways of adding type-variables α to a context: in (F-SUBTYPE) the declaration $\alpha \leq A : K$ declares α to have the kind K and to be *bounded* by the type A. In (F-TYPE) α is *unbounded* and only has a kind. This contrasts with other systems which have a "top" type $\top^K$ for each kind K, and recover unbounded type variables by assuming $\alpha \leq \top^K : K$. Since we have no direct application for top types, we steer clear of their potentially bad behaviour: it is the top types that render the subtyping relation undecidable in $F_{\leq}$ when combined with the standard contravariant rule for bounded quantifiers [23]. (In the present system, we have no type abstraction or quantification, so using top types might not invalidate our results despite adopting a contravariant rule for π-types. But we haven't investigated this).

Here are the rules for **kinding**:

$$\frac{\Gamma \vdash \star \qquad \alpha \in \mathit{Dom}(\Gamma)}{\Gamma \vdash \alpha : \mathsf{Kind}_\Gamma(\alpha)} \quad (\text{K-VAR})$$

$$\frac{\Gamma, x : A \vdash B : \star}{\Gamma \vdash \pi x{:}A.\, B : \star} \quad (\text{K-}\pi)$$

$$\frac{\Gamma, x : A \vdash B : K}{\Gamma \vdash \Lambda x{:}A.\, B : \Pi x{:}A.K} \quad (\text{K-}\Lambda)$$

$$\frac{\Gamma \vdash A : \Pi x{:}B.K \qquad \Gamma \vdash M : B}{\Gamma \vdash A\, M : K[x := M]} \quad (\text{K-APP})$$

$$\frac{\Gamma \vdash A : K \qquad \Gamma \vdash K' \qquad K =_\beta K'}{\Gamma \vdash A : K'} \quad (\text{K-CONV})$$

The rule (K-VAR) assigns a type variable α the kind given to it in the context, written $\mathsf{Kind}_\Gamma(\alpha)$. The set of variables declared in Γ is written $\mathit{Dom}(\Gamma)$.

In (K-π) the type $\pi x{:}A.\, B$ is the dependent function space. In (K-Λ) we can abstract over a type B by a term variable x to form the type family (dependent type) $\Lambda x{:}A.\, B$. Such a type function can be instantiated in the rule (K-APP). Finally the rule (K-CONV) closes the judgement under well-formed conversion of kinds.

Here are the rules for **typing**:

$$\frac{\Gamma \vdash \star \qquad x \in Dom(\Gamma)}{\Gamma \vdash x : \Gamma(x)} \tag{T-VAR}$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x{:}A.\, M : \pi x{:}A.\, B} \tag{T-λ}$$

$$\frac{\Gamma \vdash M : \pi x{:}A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]} \tag{T-APP}$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A \leq B}{\Gamma \vdash M : B} \tag{T-SUB}$$

These are completely standard. The subsumption rule (T-SUB) replaces a rule of type-conversion.

Finally, here are the rules for **subtyping**:

$$\frac{\Gamma \vdash A : K \qquad \Gamma \vdash B : K \qquad A =_\beta B}{\Gamma \vdash A \leq B} \tag{S-CONV}$$

$$\frac{\Gamma \vdash A \leq B \qquad \Gamma \vdash B \leq C}{\Gamma \vdash A \leq C} \tag{S-TRANS}$$

$$\frac{\Gamma \vdash \star \qquad \alpha \text{ bounded in } \Gamma}{\Gamma \vdash \alpha \leq \Gamma(\alpha)} \tag{S-VAR}$$

$$\frac{\Gamma \vdash A' \leq A \qquad \Gamma, x : A' \vdash B \leq B' \qquad \Gamma \vdash \pi x{:}A.\, B : \star}{\Gamma \vdash \pi x{:}A.\, B \leq \pi x{:}A'.\, B'} \tag{S-π}$$

$$\frac{\Gamma, x : A \vdash B \leq B'}{\Gamma \vdash \Lambda x{:}A.\, B \leq \Lambda x{:}A.\, B'} \tag{S-Λ}$$

$$\frac{\Gamma \vdash A \leq B \qquad \Gamma \vdash BM : K}{\Gamma \vdash A\, M \leq B\, M} \tag{S-APP}$$

Conversion is included in the subtyping relation by (S-CONV), which also ensures reflexivity on types of the same kind. Transitivity is ensured by (S-TRANS).

The rule (S-VAR) allows us to use the bound of a bounded type-variable; $\Gamma(\alpha)$ stands for the bound of α.

The subtyping rule for π-types, (S-π), is contravariant in the domain and covariant in the codomain; the codomains are compared under the stronger restriction that $x : A'$. Because of this the final judgement is needed to ensure that $\pi x{:}A.\, B$ is indeed a well-formed type.

Type families are included in the subtype relation by (S-Λ), which extends the relation pointwise. The corresponding rule for applications is (S-APP). Only families with the same

domain are comparable. (In principle it would be possible to generalise to a rule with the same form as (S-Λ), but this would break the invariant that only types of the same kind are comparable, so a relation of subkinding would be needed to compare kinds.)

Here is a brief example of using subtyping with type-dependency, to expresses basic relationships about datatypes for bags and lists. Assume that we begin with the context:

$$\begin{array}{rcll}\Gamma_{\mathit{Bag}} & \equiv & \mathit{Nat} & : \star, \\ & & \mathit{Even} \leq \mathit{Nat} & : \star, \\ & & \mathit{AllBags} & : \star, \\ & & \mathit{Bag} \leq \Lambda n{:}\mathit{Nat}.\, \mathit{AllBags} & : \Pi n{:}\mathit{Nat}.\star, \\ & & \mathit{List} \leq \mathit{Bag} & : \Pi n{:}\mathit{Nat}.\star \end{array}$$

The idea is that *AllBags* is the type of all bags, and the dependent types $\mathit{Bag}(n)$ and $\mathit{List}(n)$ represent bags and lists of size n. A list of length n is also a bag of size n.

The rule (S-π) lets us infer subtypings such as $\pi n{:}\mathit{Nat}.\, \mathit{List}(n) \leq \pi n{:}\mathit{Even}.\, \mathit{Bag}(n)$, so if we expect a function from an even number n to a bag of size n, we can use a function that maps any natural n to a list of length n.

If $n : \mathit{Nat}$, using (S-APP), (S-CONV), and (S-TRANS) we can show that $\mathit{List}(n) \leq \mathit{AllBags}$. Using (S-$\Lambda$) we can show that $\Lambda n{:}\mathit{Nat}.\, \mathit{List}(n) \leq \Lambda n{:}\mathit{Nat}.\, \mathit{Bag}(n)$, for example.

2.1 Basic properties of $\lambda P_{\leq}$

Many basic properties of $\lambda P_{\leq}$ can be established routinely, although the order of proofs is more critical than in systems without subtyping. This section contains the basic properties we need.

Proposition 2.1 (Commutativity of substitution).
If $x \not\equiv y$ and $x \notin FV(M)$ then, $(A[y := M])[x := N[y := M]] = (A[x := N])[y := M]$.

Proposition 2.2 (Church-Rosser property).
Let R be one of β_1, β_2 or β. If $U \longrightarrow\!\!\!\!\!\rightarrow_R U'$ and $U \longrightarrow\!\!\!\!\!\rightarrow_R U''$, then there exists a V such that $U' \longrightarrow\!\!\!\!\!\rightarrow_R V$ and $U'' \longrightarrow\!\!\!\!\!\rightarrow_R V$.
Proof Standard. □

Proposition 2.3 (Strong Normalization).
Let R be one of β_1, β_2 or β.

1. *If $\Gamma \vdash K$, then K is strongly R-normalizing.*
2. *If $\Gamma \vdash A : K$, then A is strongly R-normalizing.*
3. *If $\Gamma \vdash M : A$, then M is strongly R-normalizing.*

Proof To show strong normalization for $\lambda P_{\leq}$, we adapt the method given for LF in [17]. There, a reduction-preserving translation of pre-terms into Curry-typable terms of the simply-typed lambda-calculus $\lambda^{\rightarrow}$ is given, which establishes the strong normalization for LF since $\lambda^{\rightarrow}$ is known to be strongly normalizing.

To adapt this method to $\lambda P_{\leq}$, we have to make an adjustment to Definition A.9 of [17] so that the type-translation τ takes into account bounds of type variables. This reflects the fact that in LF, nothing is known about the structure of the types that a variable α ranges over, so it can be mapped to the base type ω. But in $\lambda P_{\leq}$, a variable α can be bounded, which, for example, could force it to range over Π-types. The details follow.

Definition 2.4 (Translations to $\lambda^{\rightarrow}$).
We define three translation functions on pre-terms. The function κ gives a $\lambda^{\rightarrow}$ type from a kind; it is the same as the function called τ in [17]. The function τ here gives a $\lambda^{\rightarrow}$ type from a $\lambda P_{\leq}$ type; it is given with respect to a fixed context Γ.[1] When we want to make this context explicit, we write $\tau_{\Gamma}(A)$.

$$\begin{array}{rcl} \kappa(\star) & = & \omega \\ \kappa(\Pi x{:}A.K) & = & \kappa(A) \to \kappa(K) \\ \kappa(\alpha) & = & \omega \\ \kappa(\pi x{:}A.\, B) & = & \kappa(A) \to \kappa(B) \\ \kappa(\Lambda x{:}A.\, B) & = & \kappa(B) \\ \kappa(A\,M) & = & \kappa(A) \end{array} \qquad \begin{array}{rcl} \tau(\alpha) & = & \begin{cases} \tau(A) & \text{if } \alpha \leq A : K \in \Gamma \\ \omega & \text{if } \alpha \text{ is not bounded in } \Gamma \end{cases} \\ \tau(\Lambda x{:}A.\, B) & = & \tau(B) \\ \tau(A\,M) & = & \tau(A) \\ \tau(\pi x{:}A.\, B) & = & \tau(A) \to \tau(B) \end{array}$$

We extend τ to contexts:

$$\begin{array}{rcl} \tau(\langle\rangle) & = & \langle\rangle \\ \tau(\Gamma,\, x : A) & = & \tau(\Gamma), x : \tau_{\Gamma}(A) \\ \tau(\Gamma,\, \alpha : K) & = & \tau(\Gamma), \alpha : \kappa(K) \\ \tau(\Gamma,\, \alpha \leq A : K) & = & \tau(\Gamma), \alpha : \kappa(K) \end{array}$$

The function $|\ |$ maps types and terms of $\lambda P_{\leq}$ into terms of $\lambda^{\rightarrow}$:

$$\begin{array}{rcll} |x| & = & x & \\ |\alpha| & = & \alpha & \\ |AM| & = & |A|\,|M| & \\ |MN| & = & |M|\,|N| & \\ |\pi x{:}A.\, B| & = & \mathbf{p}_{\tau(A)}\,|A|\,(\lambda x.\,|B|) & \\ |\lambda x{:}A.\, M| & = & (\lambda y.\,\lambda x.\,|M|)\,|A| & (y \notin FV(M)) \\ |\Lambda x{:}A.\, B| & = & (\lambda y.\,\lambda x.\,|B|)\,|A| & (y \notin FV(B)) \end{array}$$

The translation for π is defined using a family of simply-typed constants,

$$\mathbf{p}_{\tau} \quad : \quad \omega \to (\tau \to \omega) \to \tau$$

We consider α and x to also be variables in $\lambda^{\rightarrow}$.

Lemma 2.5 (Translation to $\lambda^{\rightarrow}$ preserves typing).
1. If $\Gamma \vdash A : K$ then $\tau(\Gamma) \vdash_{\lambda^{\rightarrow}} |A| : \kappa(K)$.

[1] We only get away with a fixed context Γ because there is no abstraction or quantification over types, so the set of bounded type variables is fixed in a typing derivation. This fact is also important to our subtyping algorithm derived later.

2. *If* $\Gamma \vdash M : A$ *then* $\tau(\Gamma) \vdash_{\lambda^{\rightarrow}} |M| : \tau_{\Gamma}(A)$.

3. *If* $\Gamma \vdash A \leq B$ *then* $\tau_{\Gamma}(A) \equiv \tau_{\Gamma}(B)$.

Proof By induction on derivations. The first two cases are similar to the proof in [17] for LF; for (T-SUB) we use the third case. The third case is easily seen, using the simple Lemma A.10 in [17] (notice that this holds for β_1, β_2 or full β conversion); the use of the bound for α in the definition of $\tau_{\Gamma}(\alpha)$ is crucial for (S-VAR). □

Proof of Proposition 2.3 Follows from Lemma 2.5, by noticing that the $|\ |$ translation preserves reductions. See [17]. □

The next proposition says that reduction commutes with substitution and that β-equality can be factored into the β_1-equality of β_2 normal forms. The facts mentioned are simply those which we need later on in the paper.

Proposition 2.6 (Reduction and conversion).

1. *If* $U \longrightarrow_{\beta_1} V$, *then* $U[x := M] \longrightarrow_{\beta_1} V[x := M]$ *and* $U[\alpha := A] \longrightarrow_{\beta_1} V[\alpha := A]$.

2. *If* $U =_{\beta_1} V$, *then* $U[x := M] =_{\beta_1} V[x := M]$ *and* $U[\alpha := A] =_{\beta_1} V[\alpha := A]$.

3. *If* $U \longrightarrow_{\beta_2} V$, *then* $U[x := M] \longrightarrow_{\beta_2} V[x := M]$ *and* $U[\alpha := A] \longrightarrow_{\beta_2} V[\alpha := A]$.

4. *If* $U =_{\beta_2} V$, *then* $U[x := M] =_{\beta_2} V[x := M]$ *and* $U[\alpha := A] =_{\beta_2} V[\alpha := A]$.

5. $(U[x := M])^{\beta_2} \equiv U^{\beta_2}[x := M^{\beta_2}]$.

6. *If* A^{β_2} *and* B^{β_2} *exist then* $A =_{\beta} B$ *implies* $A^{\beta_2} =_{\beta_1} B^{\beta_2}$.

Proof Items 1, 3 and 5 follow by induction on the structure of U. Items 2 and 4, by induction on the definition of $=_{\beta_1}$ and $=_{\beta_2}$ respectively. Item 6 uses the Church-Rosser property, Proposition 2.2. □

The next proposition concerns the behaviour of well-formed contexts. A context Γ is a *prefix* of Γ' if Γ' extends Γ by zero or more declarations. A context Γ is included in a context Γ', $\Gamma \subseteq \Gamma'$, if every declaration in Γ is also a declaration in Γ'. The size of a derivation of $\Gamma \vdash J$ is indicated by $\mathit{sizederiv}(\Gamma \vdash J)$, which we take to mean the number of rule applications used in the derivation tree; this is the measure we refer to when talking of a "shorter" derivation.

Proposition 2.7 (Context properties).

1. Variables. *If* $\Gamma \vdash J$ *then each type or term variable is declared at most once in* Γ, *and* $FV(J) \subseteq Dom(\Gamma)$.

2. Generation.

   (a) *If* $\Gamma_1, x : A, \Gamma_2 \vdash J$ *then* $\Gamma_1 \vdash A : \star$

   (b) *If* $\Gamma_1, \alpha : K, \Gamma_2 \vdash J$ *then* $\Gamma_1 \vdash K$

(c) *If* $\Gamma_1, \alpha \leq A : K, \Gamma_2 \vdash J$ *then* $\Gamma_1 \vdash A : K$

Moreover, there exists a derivation of the consequent which is shorter than the derivation of the antecendent.

3. Well-formedness.
Suppose $\Gamma \vdash J$. *Then for every prefix* Γ' *of* Γ, *we have* $\Gamma' \vdash \star$.
Moreover, if $J \not\equiv \star$ *or* Γ' *is a proper prefix of* Γ, *then* $\mathsf{sizederiv}(\Gamma' \vdash \star) < \mathsf{sizederiv}(\Gamma \vdash J)$.

4. Renaming.
Suppose θ *is a mapping from variables to variables. Then* $\Gamma \vdash J$ *implies* $\theta(\Gamma) \vdash \theta(J)$, *where* $\theta(-)$ *denotes the obvious extensions of the mapping.*

5. Thinning.
Suppose $\Gamma \subseteq \Gamma'$, $\Gamma \vdash J$ *and* $\Gamma' \vdash \star$. *Then* $\Gamma' \vdash J$.

Substitution holds for each sort of variable assumption.

Proposition 2.8 (Substitution).

1. *If* $\Gamma_1, x : A, \Gamma_2 \vdash J$ *and* $\Gamma_1 \vdash M : A$, *then* $\Gamma_1, \Gamma_2[x := M] \vdash J[x := M]$.
2. *If* $\Gamma_1, \alpha : K, \Gamma_2 \vdash J$ *and* $\Gamma_1 \vdash A : K$, *then* $\Gamma_1, \Gamma_2[\alpha := A] \vdash J[\alpha := A]$.
3. *If* $\Gamma_1, \alpha \leq B : K, \Gamma_2 \vdash J$ *and* $\Gamma_1 \vdash A \leq B$, *then* $\Gamma_1, \Gamma_2[\alpha := A] \vdash J[\alpha := A]$.

Proof Routine. Each part by simultaneous induction on derivations for the four judgement forms. □

One desirable property of a type system is *type unicity*: the type of a term is unique up to conversion. With subtyping this cannot hold, although we can hope for the property of *minimal types*. This property is useful because it allows us to factor the problem of type-checking into two parts: the inference of a minimal type for a term and deciding the subtyping relation. We will prove that $\lambda P_{\leq}$ has minimal types in Section 4.

For the kinding fragment of our system, however, unicity does hold. The next proposition is that the kind of a type is unique up to conversion. We use the observation that conversion at the kind level is particularly simple since there is no application. If $K =_R K'$ (where R is one of β, β_1, β_2) then for some $n \geq 0$, $K \equiv \Pi x{:}A_1 \ldots \Pi x{:}A_n.\star$ and $K' \equiv \Pi x{:}A'_1 \ldots \Pi x{:}A'_n.\star$ with $A_i =_R A'_i$ for each i.

Proposition 2.9 (Unicity of kinds).
If $\Gamma \vdash A : K_1$ *and* $\Gamma \vdash A : K_2$, *then* $K_1 =_\beta K_2$.

Proof By induction on the sum of the heights of the derivation of $\Gamma \vdash A : K_1$ and of $\Gamma \vdash A : K_2$. If either derivation ends in (K-CONV), the result follows immediately from the induction hypothesis and transitivity of $=_\beta$. Otherwise, we consider the last rule in each derivation, which must be the same. We show here the case for (K-APP), when $A \equiv A_1 M$. We have $\Gamma \vdash A_1 : \Pi x{:}B_1.K_1$ and $\Gamma \vdash A_1 : \Pi x{:}B_2.K_2$. Using the induction hypothesis, $K_1 =_\beta K_2$. Since conversion is preserved by substitution (Proposition 2.6(2,4)), $K_1[x := M] =_\beta K_2[x := M]$ as required. □

Bound narrowing is the name given to the property that derivability of a judgement is preserved by replacing the bounding type in a type-variable declaration by a type which is smaller in the subtype relation. Informally, one can see this is true by adding an instance of subsumption or transitivity to each use of a variable rule. (And so the derivation of the judgement with a narrowed context may be longer than the original one).

We first prove a restricted form of this property.

Proposition 2.10 (Bound narrowing I).
Suppose $\Gamma \vdash A' \leq A$. *Then*

1. $\Gamma, x : A, \Gamma' \vdash J$ *and* $\Gamma \vdash A' : \star$ *implies* $\Gamma, x : A', \Gamma' \vdash J$
2. $\Gamma, \alpha \leq A : K, \Gamma' \vdash J$ *and* $\Gamma \vdash A' : K$ *implies* $\Gamma, \alpha \leq A' : K, \Gamma' \vdash J$

Proof

1. We prove the statement simultaneously for the four judgement forms, for all Γ_1 and Γ_2, by induction on derivations. For (F-TERM), we use the assumption $\Gamma \vdash A' : \star$ and Proposition 2.7; for (T-VAR) when the variable being typed is x we also use (T-SUB). The remaining cases are straightforward.
2. Similar to 1. For formation, we use the assumption and Proposition 2.7 to show $\Gamma \vdash A' : K$ in (F-SUBTYPE). For subtyping derived with (S-VAR), we must show $\Gamma' \vdash \alpha \leq A$, which follows via (S-TRANS), using Proposition 2.7 and the assumption. Remaining cases are straightforward. □

The next property shows some anticipated agreements between the judgements, for example, that every type inhabited by a term indeed has kind $\star$.

Proposition 2.11 (Agreement of judgements).

1. *If* $\Gamma \vdash A : K$ *then* $\Gamma \vdash K$.
2. *If* $\Gamma \vdash M : A$ *then* $\Gamma \vdash A : \star$.
3. *If* $\Gamma \vdash A \leq B$ *then* $\Gamma \vdash A, B : K$, *for some* K.

Proof By induction on derivations; parts 2 and 3 are proved together. We use Proposition 2.8 for (K-APP), (T-APP) and (S-APP); Proposition 2.7 for (T-VAR) and (S-VAR), and Proposition 2.10 for (S-π) and (S-Λ). □

Agreement has important consequences. For example, we can see that the usual λP rule of conversion for typing is admissible:

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A' : \star \qquad A =_\beta A'}{\Gamma \vdash M : A'} \qquad \text{(T-CONV)}$$

Using Proposition 2.11, $\Gamma \vdash A : \star$ is implied by the first premise. So $\Gamma \vdash A \leq A'$ using the second and third premises with (S-CONV). Then $\Gamma \vdash M : A'$ follows using subsumption, (T-SUB).

We can also use agreement to get a stronger version of Proposition 2.10. We write $\Gamma \vdash J$ to denote an arbitrary judgement.

Proposition 2.12 (Bound narrowing II).
Suppose $\Gamma \vdash A' \leq A$. *Then*

1. $\Gamma, x : A, \Gamma' \vdash J$ *implies* $\Gamma, x : A', \Gamma' \vdash J$
2. $\Gamma, \alpha \leq A : K, \Gamma' \vdash J$ *implies* $\Gamma, \alpha \leq A' : K, \Gamma' \vdash J$

Proof From Proposition 2.10. Using context properties, Proposition 2.9 and Proposition 2.11 we get $\Gamma_1 \vdash A' : \star$ in part 1 and $\Gamma_1 \vdash A' : K$ in part 2. □

2.2 Towards subject reduction

Another desirable property for type systems is *subject reduction*. This is the property that β-reduction preserves the type of a term. (Since a term may have several types in a subtyping system, and since an abstraction term $\lambda x{:}A.\, M$ may be applied to a term whose minimal type is smaller than A, in general we may have that reduction adds types.)

To prove subject reduction we need to reason about the way judgements are derived. This is the point where we hit a snag. In particular, to show that $(\lambda x{:}A.\, M)\, N$ and its reduct $M[x := N]$ have the same type, we would like to assume that the application was typed using (T-λ) followed by (T-APP). For this we need a *generation principle*.

Proposition 2.13 (Generation for typing).

1. *If* $\Gamma \vdash x : C$ *then* $\Gamma \vdash \Gamma(x) \leq C$.
2. *If* $\Gamma \vdash \lambda x{:}A.\, M : C$ *then for some* B,
   (a) $\Gamma, x : A \vdash M : B$ *and (b)* $\Gamma \vdash \pi x{:}A.\, B \leq C$.
3. *If* $\Gamma \vdash MN : C$ *then for some* A, B,
   (a) $\Gamma \vdash M : \pi x{:}A.\, B$, *(b)* $\Gamma \vdash N : A$, *and (c)* $\Gamma \vdash B[x := N] \leq C$.

Proof By induction on typing derivations, using transitivity of subtyping. □

However, this is too weak to show type preservation; the possibility that subtyping was used in (T-SUB) gets in the way. Suppose $(\lambda x{:}A.\, M)N : C$. We want to show that $\Gamma \vdash M[x := N] : C$ as well. By generation for typing, for some A_1 and B_1:

$$\Gamma \vdash (\lambda x{:}A.\, M) : \pi x{:}A_1.\, B_1, \quad \Gamma \vdash N : A_1 \quad \text{and} \quad \Gamma \vdash B_1[x := N] \leq C.$$

Again, by generation for typing, for some B_2

$$\Gamma, x : A \vdash M : B_2 \quad \text{and} \quad \Gamma \vdash \pi x{:}A.\, B_2 \leq \pi x{:}A_1.\, B_1.$$

If we could show that

$$\Gamma \vdash A_1 \leq A \quad \text{and} \quad \Gamma, x : A_1 \vdash B_2 \leq B_1$$

then we could continue as follows. By narrowing,

$$\Gamma, x : A_1 \vdash M : B_2$$

then, by (T-SUB),

$$\Gamma, x : A_1 \vdash M : B_1,$$

and, by the substitution property, Proposition 2.8,

$$\Gamma \vdash M[x := N] : B_1[x := N].$$

Finally, using (T-SUB),

$$\Gamma \vdash M[x := N] : C.$$

And that would be it. The judgements $\Gamma \vdash A_1 \leq A$ and $\Gamma, x : A_1 \vdash B_2 \leq B_1$ are the problem; we would hope to prove them using a generation property for subtyping, applied to $\Gamma \vdash \pi x{:}A.\, B_2 \leq \pi x{:}A_1.\, B_1$. Unfortunately, we cannot prove a suitable generation principle directly by induction on subtyping derivations because of the rules (S-CONV) and (S-TRANS). The next section is a quest towards a generation principle for subtyping using a formulation without these troublesome rules.

3 A subtyping algorithm

To delve further into the meta-theory of $\lambda\mathrm{P}_{\leq}$ we must confront the subtyping system. We do this by analysing an equivalent system which is *syntax directed* (to derive any given statement, at most one rule applies), and so forms an algorithm when viewed in reverse. A generation principle for a syntax-directed system is immediate; the hard part is proving its equivalence with the original presentation.

Our algorithmic presentation is akin to that for $F^{\omega}_{\wedge}$ in [13], with two important differences. First, the rules here have no kinding premises, so the cycle of dependencies between subtyping and typing is destroyed. Second, we make a novel adjustment for dependent types: splitting β-reduction.

We shall explain the reason for splitting β-reduction shortly. Why remove kinding premises from the subtyping rules? This was a technique used in the study of $F^{\omega}_{\leq}$ in [26], but we know from the $F^{\omega}_{\wedge}$ algorithm in [13] that removing kinding is not crucial to the study of that system. Things are more complex with $\lambda\mathrm{P}_{\leq}$ because of the circularity between typing and subtyping: keeping kinding premises, we could reduce deciding $\Gamma \vdash_{\mathcal{A}} A \leq B$ to a finite number of typing constraints, but such constraints are in no obvious way "smaller" than the subtyping statement we began with. So it is hard to argue that an algorithm cannot loop by an infinite alternation of calls from one judgement to the other. Our first plan was to seek a cunning induction measure, but removing the circularity seems conceptually simpler and moreover closer to a practical subtyping algorithm.

The new rules derive statements $\Gamma \vdash_{\mathcal{A}} A \leq B$, with A and B in β_2-normal form. Normal forms allow us to grasp the fine structure of the subtyping relation, since occurrences of applications are restricted. Otherwise it is hard to tell whether an occurrence of $A\,M$ was introduced by (S-APP) or (S-CONV), for example.

There are four rules definining the algorithmic subtyping relation.

$$\frac{\Gamma \vdash_{\mathcal{A}} A' \leq A \qquad \Gamma, x : A' \vdash_{\mathcal{A}} B \leq B'}{\Gamma \vdash_{\mathcal{A}} \pi x{:}A.\, B \leq \pi x{:}A'.\, B'} \quad (\text{AS-}\pi)$$

$$\frac{A =_{\beta_1} A' \qquad \Gamma, x : A' \vdash_{\mathcal{A}} B \leq B'}{\Gamma \vdash_{\mathcal{A}} \Lambda x{:}A.\, B \leq \Lambda x{:}A'.\, B'} \quad (\text{AS-}\Lambda)$$

$$\frac{M_1 =_{\beta_1} M_1' \ \cdots \ M_n =_{\beta_1} M_n'}{\Gamma \vdash_{\mathcal{A}} \alpha\, M_1 \cdots M_n \ \leq \ \alpha\, M_1' \cdots M_n'} \quad (\text{AS-APP-R})$$

$$\frac{\begin{array}{l}\alpha \text{ bounded in } \Gamma,\ A \neq_{\beta_1} \alpha\, M_1 \cdots M_n \\ \Gamma \vdash_{\mathcal{A}} (\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2} \leq A\end{array}}{\Gamma \vdash_{\mathcal{A}} \alpha\, M_1 \cdots M_n \leq A} \quad (\text{AS-APP-T})$$

The first two rules correspond to (S-π) and (S-Λ), except that the kinding premises are removed and β_1-conversion of the type-label of Λ is allowed. The two rule schemes for application guarantee that the algorithmic subtyping relation is closed under reflexivity and transitivity (this claim is proved in Section 3.3). In the rule scheme (AS-APP-R), "R" stands for reflexivity and in (AS-APP-T), "T" stands for transitivity. To make the rules syntax-directed, we need the premise in (AS-APP-T) that $A \neq_{\beta_1} \alpha\, M_1 \cdots M_n$, otherwise (AS-APP-R) would apply.

The motivation to use β_2-normal forms instead of full β-normal forms appears when designing (AS-APP-T). To make the new system deterministic, we must remove the transitivity rule. However, it cannot be eliminated completely so it is restricted: we only allow transitivity along the bound of a type-variable in head position of a normal form. To check whether $\Gamma \vdash_{\mathcal{A}} \alpha\, M_1 \cdots M_n \leq A$, we check if $\Gamma \vdash_{\mathcal{A}} (\Gamma(\alpha)\, M_1 \cdots M_n) \leq A$. But this step introduces a possibly non-normal form, so the algorithm must normalize $\Gamma(\alpha)\, M_1 \cdots M_n$. As a first attempt, we get the rule:

$$\frac{\Gamma \vdash_{\mathcal{A}} (\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta} \leq A}{\Gamma \vdash_{\mathcal{A}} \alpha\, M_1 \cdots M_n \leq A}$$

Because the algorithmic rules do not check kinding, we must ensure that if we start with well-kinded types in the conclusion (the arguments, seen as an algorithm), we still have well-kinded types in the hypothesis (the arguments in any recursive call). Starting with $\Gamma \vdash_{\mathcal{A}} \alpha\, M_1 \cdots M_n : K$, we can prove (using Proposition 3.3 below) that replacing α by its bound preserves kinding, so $\Gamma \vdash_{\mathcal{A}} (\Gamma(\alpha)\, M_1 \cdots M_n) : K$ too. Now we need to prove that

$$\Gamma \vdash_{\mathcal{A}} (\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta} : K$$

but this is proved using the subject reduction property for β, exactly the result that we could not prove without an algorithm for subtyping. So we are back where we started.

Fortunately, we can recover from this using β_2-normal forms instead of β-normal forms. To see the structure of types, β_2-normalization is enough, and subject reduction for β_2-reduction can be proved easily. This explains the use of β_2-reduction in (AS-APP-T).

The other rule that makes the original subtyping system non syntax-directed is (S-CONV). Since we use types in β_2-normal form, the rule for reflexivity must incorporate β_1-conversion, and it suffices to use reflexivity on terms of the form $\alpha\, M_1 \cdots M_n$ for $n \geq 0$. This explains (AS-APP-R).

The following subsections prove that the algorithmic subtyping relation is equivalent to the original system, and apply the algorithm to prove results about the original system. In Section 3.1 we prove that the original presentation is closed under β_2-reduction. In Section 3.2 we prove that the algorithmic rules are sound for the original presentation. In Section 3.3 we prove that reflexivity and transitivity are admissible in the algorithm, which is the core of the proof following in Section 3.4 that the algorithmic rules are complete. In Section 3.5 the equivalence result is stated, and used to prove the sought after generation principle for subtyping,

3.1 Closure under β_2-reduction

To prove subject reduction for β_2-reduction, we need generation only for kinding.

Proposition 3.1 (Generation for kinding).

1. *If* $\Gamma \vdash \alpha : K$ *then* $K =_\beta \mathsf{Kind}_\Gamma(\alpha)$.
2. *If* $\Gamma \vdash \pi x{:}A.\, B : K$ *then* $K \equiv \star$, *and* $\Gamma, x : A \vdash B : \star$.
3. *If* $\Gamma \vdash \Lambda x{:}A.\, B : K$ *then there exists* K' *such that* $K =_\beta \Pi x{:}A.K'$, *and* $\Gamma, x : A \vdash B : K'$.
4. *If* $\Gamma \vdash A\, M : K$ *then there exists* B, K' *such that* $\Gamma \vdash A : \Pi x{:}B.K'$, $\Gamma \vdash M : B$ *and* $K'[x := M] =_\beta K$.

Moreover, the derivations of the consequents can be assumed to be shorter than the derivation of the antecedents.

Proof By induction on derivations. In each case, the antecedent must either have been derived by a structural rule, when the result is immediate, or by (K-CONV), when we use the induction hypothesis and transitivity of β-conversion. □

As well as β_2 subject reduction for kinding, we also need closure of the subtyping relation under β_2-reduction. So we state a generalized form of the property, writing $J \longrightarrow_{\beta_2} J'$ to indicate a β_2-reduction inside J.

Proposition 3.2 (Closure under β_2-reduction).
If $\Gamma \vdash J$ *and* $J \longrightarrow\!\!\!\!\rightarrow_{\beta_2} J'$ *then* $\Gamma \vdash J'$.

Proof The one step case follows by induction on the derivation of $\Gamma \vdash J$ also proving simultaneously the statement for a reduction inside the context Γ. The only interesting case is for an outermost reduction in the kinding judgement. Suppose the premises are $\Gamma \vdash \Lambda x{:}A.\, B : \Pi x{:}A'.K$ and $\Gamma \vdash M : A'$, and the reduction is $(\Lambda x{:}A.\, B)\, M \longrightarrow_{\beta_2} B[x := M]$. By kinding generation, Proposition 3.1, we get $\Gamma, x : A \vdash B : K'$ with $K =_\beta K'$ and $A =_\beta A'$. Using context properties, $\Gamma \vdash A : \star$ and so by (T-CONV), $\Gamma \vdash M : A$. By

substitution, $\Gamma \vdash B[x := M] : K'[x := M]$. The result $\Gamma \vdash B[x := M] : K[x := M]$ follows by preservation of β-equality under substitution (Proposition 2.6(2,4)) and (K-CONV) using agreement for the original conclusion.

The result for multiple reductions follows by induction on the definition of $\twoheadrightarrow_{\beta_2}$. The one step case is what we have just proved, the reflexivity case is immediate and the transitivity case is by the induction hypothesis. □

3.2 Soundness

In the soundness proof, β_2 subject reduction for kinding is crucial for the case of (AS-APP-T) to show that kindability is preserved from the conclusion to the premise.

The soundness lemma requires an auxiliary proposition.

Proposition 3.3 (Bounded type variables).

1. *If* $\Gamma \vdash \alpha\, M_1 \cdots M_n : K$, *then* $\Gamma \vdash \Gamma(\alpha)\, M_1 \cdots M_n : K$.
2. *If* $\Gamma \vdash \alpha\, M_1 \cdots M_n : K$, *then* $\Gamma \vdash \alpha\, M_1 \cdots M_n \leq \Gamma(\alpha)\, M_1 \cdots M_n$.

Proof

1. By induction on the derivation of $\Gamma \vdash \alpha\, M_1 \cdots M_n : K$ using structural properties.
2. By induction on n, using structural properties and (S-VAR) in the base case, part 1 and (S-APP) in the inductive step. □

We can now prove soundness by a straightforward induction.

Lemma 3.4 (Soundness of algorithmic subtyping).
Suppose we have two types of the same kind, $\Gamma \vdash A, B : K$. *Then* $\Gamma \vdash_{\mathcal{A}} A \leq B$ *implies* $\Gamma \vdash A \leq B$.

Proof By induction on the derivation in the algorithmic system.

Case (AS-Λ): Suppose the conclusion is $\Gamma \vdash_{\mathcal{A}} \Lambda x{:}A_1.\, B_1 \leq \Lambda x{:}A_2.\, B_2$. We split the problem into two steps. We first prove that $\Gamma \vdash \Lambda x{:}A_2.\, B_1 \leq \Lambda x{:}A_2.\, B_2$ using the induction hypothesis; second we prove $\Gamma \vdash \Lambda x{:}A_1.\, B_1 \leq \Lambda x{:}A_2.\, B_1$ using (S-CONV). The result then follows by transitivity.

First step. To use the induction hypothesis on $\Gamma, x : A_2 \vdash_{\mathcal{A}} B_1 \leq B_2$ we need to show that $\Gamma, x : A_2 \vdash B_1, B_2 : K'$ for some K'. By the assumption and Proposition 3.1, for some K_1, K_2 we have $\Gamma, x : A_1 \vdash B_1 : K_1$ and $\Gamma, x : A_2 \vdash B_2 : K_2$ with $K =_\beta \Pi x{:}A_1.K_1 =_\beta \Pi x{:}A_2.K_2$. So by the Church-Rosser property, $K_1 =_\beta K_2$. By Proposition 2.7, $\Gamma \vdash A_1, A_2 : \star$, and, by (S-CONV), $\Gamma \vdash A_2 \leq A_1$. By narrowing, Proposition 2.12, $\Gamma, x : A_2 \vdash B_1 : K_1$, and by agreement, Proposition 2.11, $\Gamma, x : A_2 \vdash K_2$ and, by (K-CONV), $\Gamma, x : A_2 \vdash B_1 : K_2$. Take K' to be K_2. We can now apply the induction hypothesis and infer $\Gamma, x : A_2 \vdash B_1 \leq B_2$, and, by (S-Λ), $\Gamma \vdash \Lambda x{:}A_2.\, B_1 \leq \Lambda x{:}A_2.\, B_2$.

Second step. By (K-Λ), $\Gamma \vdash \Lambda x{:}A_1.\, B_1 : \Pi x{:}A_1.K_1$ and $\Gamma \vdash \Lambda x{:}A_2.\, B_1 : \Pi x{:}A_2.K_1$. By agreement, Proposition 2.11, $\Gamma \vdash \Pi x{:}A_1.K_1$. By (K-CONV), $\Gamma \vdash \Lambda x{:}A_2.\, B_1 : \Pi x{:}A_1.K_1$. Finally, by (S-CONV), $\Gamma \vdash \Lambda x{:}A_1.\, B_1 \leq \Lambda x{:}A_2.\, B_1$.

Case (AS-π): Suppose the conclusion is $\Gamma \vdash_{\mathcal{A}} \pi x{:}A_1.\, B_1 \leq \pi x{:}A_2.\, B_2$. Then by the assumption and Proposition 3.1 we have $\Gamma, x : A_1 \vdash B_1 : \star$ and $\Gamma, x : A_2 \vdash B_2 : \star$. By Proposition 2.7, $\Gamma \vdash A_1, A_2 : \star$, and by the induction hypothesis, $\Gamma \vdash A_2 \leq A_1$. By narrowing, Proposition 2.12, $\Gamma, x : A_2 \vdash B_1 : \star$ and so by the induction hypothesis, $\Gamma, x : A_2 \vdash B_1 \leq B_2$. The result now follows via (S-π) using the second assumption.

Case (AS-APP-R): Immediately using (S-CONV).

Case (AS-APP-T): where $A \equiv \alpha\, M_1 \ldots M_n$, we use Proposition 3.3(1) and subject β_2-reduction to show that $\Gamma \vdash (\Gamma(\alpha)\, M_1 \ldots M_n)^{\beta_2} : K$, hence by the induction hypothesis, $\Gamma \vdash (\Gamma(\alpha)\, M_1 \ldots M_n)^{\beta_2} \leq A$. Now by Proposition 3.3(2), $\Gamma \vdash \alpha\, M_1 \ldots M_n \leq \Gamma(\alpha)\, M_1 \ldots M_n$. The result follows using (S-CONV) and (S-TRANS) twice. □

The proof above shows that each derivation in the algorithmic system induces a derivation in the original system; in effect, the algorithm suggests a strategy for using the rules of the original system. A derivation in the algorithm induces a derivation in the original system which uses transitivity only on variables, if at all, and which uses conversion only at the beginning and when subtyping applications.

3.3 Reflexivity and transitivity

For completeness we first show that reflexivity and transitivity are admissible in the new system. This is like the cut-elimination argument first used in a subtyping setting by Curien and Ghelli [16] for their study of $F_{\leq}$. But instead of showing that reflexivity and transitivity can be removed, we show that they can be added without changing the derivable statements. This avoids consideration of special "cut-free" derivations.

Proposition 3.5 (Reflexivity of algorithmic subtyping).
Let A and A' be two types in β_2-normal form, with $A =_{\beta_1} A'$ and $\Gamma \vdash A, A' : K$. Then $\Gamma \vdash_{\mathcal{A}} A \leq A'$.

Proof By induction on $\mathsf{size}(A) + \mathsf{size}(A')$, where $\mathsf{size}(U)$ is the number of symbols in U. Since A and A' are in β_2-normal form, they may only differ at term components, so we consider four cases.

Case $A \equiv A' \equiv \alpha$: By (AS-APP-R).

Case $A \equiv \pi x{:}B.\, C$, $A' \equiv \pi x{:}B'.\, C'$: Using the induction hypothesis and (AS-π).

Case $A \equiv \Lambda x{:}B.\, C$, $A' \equiv \Lambda x{:}B'.\, C'$: Using the induction hypothesis and (AS-Λ).

Case $A \equiv B\, M$, $A' \equiv B'\, M'$: Since A and A' are in β_2-normal form, we must have $B \equiv \alpha M_1 \ldots M_n$ and $B' \equiv \alpha M'_1 \ldots M'_n$ for some $n \geq 0$, with $M_i =_{\beta_1} M'_i$. The result follows immediately by (AS-APP-R). □

Showing admissibility of transitivity uses extra machinery. To define a measure for the main induction, we extend the language with a new type constructor and a new reduction. The crucial property of the measure is that it reduces from the conclusion to the premises of the algorithmic subtyping rules, notably (AS-APP-T). The same measure will be used to show termination of the subtyping algorithm.

The new type constructor is a binary "plus" operator, which has the kinding rule:

$$\frac{\Gamma \vdash A : K \qquad \Gamma \vdash B : K}{\Gamma \vdash A + B : K} \tag{K-+}$$

The idea is this. Subtyping bounded type variables α typically, but not necessarily, can involve using transitivity along the bound: $\alpha \leq \Gamma(\alpha) \leq D$. A type thus contains many "choice" points where the bound of a variable may or may not be used during subtyping. We define an operation $\mathit{plus}_\Gamma(C)$ which expands these points by recursively replacing bounded variables α in a type C with $\alpha + \Gamma(\alpha)$.

We can recover a plus-free type from $\mathit{plus}_\Gamma(C)$ by choosing either the left or right side of every plus expression. This is captured by +-reduction:

$$\begin{array}{lll} \mathbf{C}[A + B] & \longrightarrow_+ & \mathbf{C}[A] \\ \mathbf{C}[A + B] & \longrightarrow_+ & \mathbf{C}[B] \end{array}$$

(where $\mathbf{C}[-]$ is a type or term in the extended language with a hole in it). The number of +-reductions possible from $\mathit{plus}_\Gamma(C)$ affects the complexity of deciding a subtyping statement containing the type C.

Definition 3.6 (Plus-expansion of a type).
Let Γ be a context and declare all the type variables of a type C. Then $\mathit{plus}_\Gamma(C)$ is given by:

$$\begin{array}{rcl} \mathit{plus}_{\Gamma_1, \alpha \leq A:K, \Gamma_2}(\alpha) & = & \alpha + \mathit{plus}_{\Gamma_1}(A) \\ \mathit{plus}_{\Gamma_1, \alpha:K, \Gamma_2}(\alpha) & = & \alpha \\ \mathit{plus}_\Gamma(\pi x{:}A.\, B) & = & \pi x{:}\, \mathit{plus}_\Gamma(A).\ \mathit{plus}_\Gamma(B) \\ \mathit{plus}_\Gamma(\Lambda x{:}A.\, B) & = & \Lambda x{:}\, \mathit{plus}_\Gamma(A).\ \mathit{plus}_\Gamma(B) \\ \mathit{plus}_\Gamma(A\, M) & = & \mathit{plus}_\Gamma(A)\, M \end{array}$$

When the condition on Γ is met, $\mathit{plus}_\Gamma(C)$ is defined uniquely — this can be shown by appealing to properties of contexts and observing that the definition is well-founded on the lexicographic ordering of pairs $\langle \mathit{length}(\Gamma), \mathit{size}(C) \rangle$, where $\mathit{length}(\Gamma)$ is the number of variables declared by Γ.

One important fact is that there is a +-reduction from the expansion of a type-variable to its bound in the context; this is used in the next proposition. We write $\longrightarrow\!\!\!\!\rightarrow_R^n$ to indicate that a reduction is n steps long and $\longrightarrow\!\!\!\!\rightarrow_R^{>n}$ for more than n steps. We extend β_2-reduction to $A + B$ in the obvious (compatible) way.

Proposition 3.7 (Plus types and reduction).
$\mathit{plus}_\Gamma(\alpha\, M_1 \cdots M_n) \longrightarrow\!\!\!\!\rightarrow_{\beta_2+}^{>0} \mathit{plus}_\Gamma((\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2})$.

Proof We use several sub-lemmas to prove the statement:

1. For two contexts Γ and Γ', if $\Gamma \subseteq \Gamma'$ then $\mathsf{plus}_\Gamma(A) = \mathsf{plus}_{\Gamma'}(A)$.
2. $\mathsf{plus}_\Gamma(\alpha\, M_1 \cdots M_n) \longrightarrow\!\!\!\!\!\rightarrow_+^{>0} \mathsf{plus}_\Gamma(\Gamma(\alpha)\, M_1 \cdots M_n)$.
3. If $x \notin Dom(\Gamma)$ and $FV(A) \in Dom(\Gamma) - \{x\}$ then $\mathsf{plus}_\Gamma((A[x := M])) = (\mathsf{plus}_\Gamma(A))[x := M]$;
4. If $A \longrightarrow_{\beta_2} B$, then $\mathsf{plus}_\Gamma(A) \longrightarrow\!\!\!\!\!\rightarrow_{\beta_2} \mathsf{plus}_\Gamma(B)$.

Parts 1 and 3 follow by induction on the structure of A. Part 2 follows by induction on n: in the base case, we have $\mathsf{plus}_\Gamma(\alpha) \longrightarrow\!\!\!\!\!\rightarrow_+^{>0} \mathsf{plus}_\Gamma(\Gamma(\alpha))$ by the definition of plus. Part 4 follows by induction on the structure of A using 3. The desired result then follows from 2 and 4. □

Now β_2+-reduction will help define the measure we seek. First, let $\mathsf{maxred}_\Gamma(A)$ be the maximal number of β_2+-reductions from the plus-expansion of a type:

$$\mathsf{maxred}_\Gamma(A) =_{\text{def}} \quad \mathsf{max}\left\{ n \;\middle|\; \mathsf{plus}_\Gamma(A) \longrightarrow\!\!\!\!\!\rightarrow_{\beta_2+}^{n} A' \text{ for some } A' \right\}$$

(Notice that $\mathsf{maxred}_\Gamma(A)$ only makes sense when $\mathsf{plus}_\Gamma(A)$ is β_2+-strongly normalizing.) Then we define the *weight* of two types A, B as the pair:

$$\mathsf{weight}_\Gamma(A, B) =_{\text{def}} \quad \langle\, \mathsf{maxred}_\Gamma(A) + \mathsf{maxred}_\Gamma(B),\, \mathsf{size}(A) + \mathsf{size}(B) \,\rangle$$

The number of bounded variables and the size of the types both contribute. Pairs $\mathsf{weight}_\Gamma(A, B)$ are well-ordered by the usual lexicographic ordering.

We now mention the extension of strong normalization to the system with the + constructor necessary to show that $\mathsf{maxred}_\Gamma(A)$ is defined whenever A is well-kinded with respect to Γ.

Proposition 3.8 (Normalization with +).

1. *Strong normalization (Proposition 2.3) also holds for the +-enriched language with* (K-+) *and* β+ *reduction.*
2. *If* $\Gamma \vdash A : K$, *then* $\Gamma \vdash \mathsf{plus}_\Gamma(A) : K$ *in the + enriched language.*

Proof We can prove part 1 by an extension of the proof for Proposition 2.3, using a version of the simply-typed lambda-calculus extended with a +-reduction as in [13]. Part 2 is proved by induction on the derivation of $\Gamma \vdash A : K$. □

The following lemmas are used in the proof of transitivity admissibility.

The first is a generalized version of a bound narrowing result for typing assumptions in algorithmic subtyping; intuitively the subtyping rules ignore their typing assumptions. If this lemma seems surprising, remember that the algorithmic subtyping rules make no check on the well-formedness of the context, so valid judgements may contain non well-kinded types.

Lemma 3.9 (Bound change).
If $\Gamma, x : A, \Gamma' \vdash_{\mathcal{A}} B \leq B'$ *then* $\Gamma, x : A', \Gamma' \vdash_{\mathcal{A}} B \leq B'$

Proof By induction on the derivation of $\Gamma, x : A, \Gamma' \vdash_{\mathcal{A}} B \leq B'$ □

Lemma 3.10.

1. *If* $\Gamma \vdash \alpha\, M_1 \cdots M_n : K$, *then* $\Gamma \vdash \alpha\, M_1 \cdots M_n \leq (\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2}$.
2. *If* $\Gamma \vdash \alpha\, M_1 \cdots M_n : K$, *then* $\Gamma \vdash (\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2} : K$.

Proof Item 1 follows from Proposition 3.3 and Proposition 3.2. Item 2 follows from 1 using agreement, Proposition 2.11, and unicity of kinds, Proposition 2.9. □

Proposition 3.11 (Transitivity of algorithmic subtyping).
Let $\Gamma \vdash A, B, C : K$. *Then* $\Gamma \vdash_{\mathcal{A}} A \leq B$ *and* $\Gamma \vdash_{\mathcal{A}} B \leq C$ *implies* $\Gamma \vdash_{\mathcal{A}} A \leq C$.

Proof For all Γ using induction on $\mathsf{weight}_\Gamma(A, C)$. Since A, B, and C are well-kinded in Γ, we have that $\mathsf{plus}_\Gamma(A)$, $\mathsf{plus}_\Gamma(B)$, $\mathsf{plus}_\Gamma(C)$ and the plus-expansion of the bound of every type variable in Γ are all defined and β_2+-strongly normalizing. Therefore the inductive measure is always well-defined. Then using case analysis on the last rule used to derive $\Gamma \vdash_{\mathcal{A}} A \leq B$ we can break down the transitivity into smaller instances, using Proposition 3.7.

Case (AS-Π): From:

$$\frac{\Gamma \vdash_{\mathcal{A}} A_2 \leq A_1 \qquad \Gamma, x : A_2 \vdash_{\mathcal{A}} B_1 \leq B_2}{\Gamma \vdash_{\mathcal{A}} \pi x{:}A_1.\, B_1 \leq \pi x{:}A_2.\, B_2}$$

and:

$$\frac{\Gamma \vdash_{\mathcal{A}} A_3 \leq A_2 \qquad \Gamma, x : A_3 \vdash_{\mathcal{A}} B_2 \leq B_3}{\Gamma \vdash_{\mathcal{A}} \pi x{:}A_2.\, B_2 \leq \pi x{:}A_3.\, B_3}$$

Since there is no kinding information in the algorithm, the kindedness of the subexpressions of the types we started from has to be obtained by structural properties of the original system, using the kinding assumptions of the present proposition. By generation (Proposition 3.1), it follows that

$$\begin{array}{rl} K \equiv \star, & \Gamma, x : A_1 \vdash B_1 : \star, \\ & \Gamma, x : A_2 \vdash B_2 : \star, \quad \text{and} \\ & \Gamma, x : A_3 \vdash B_3 : \star, \end{array}$$

and, by agreement (Proposition 2.11), $\Gamma \vdash A_1, A_2, A_3 : \star$.

We can now apply the induction hypothesis to get:

$$\frac{\Gamma \vdash_{\mathcal{A}} A_3 \leq A_2 \qquad \Gamma \vdash_{\mathcal{A}} A_2 \leq A_1}{\Gamma \vdash_{\mathcal{A}} A_3 \leq A_1}$$

By narrowing (Proposition 2.12)

$$\Gamma, x : A_3 \vdash B_1 : \star \quad \text{and} \quad \Gamma, x : A_3 \vdash B_2 : \star$$

and we use the bound change lemma (Lemma 3.9) to obtain $\Gamma, x : A_3 \vdash_{\mathcal{A}} B_1 \leq B_2$. Then we can apply the induction hypothesis again to get:

$$\frac{\Gamma, x : A_3 \vdash_{\mathcal{A}} B_1 \leq B_2 \qquad \Gamma, x : A_3 \vdash_{\mathcal{A}} B_2 \leq B_3}{\Gamma, x : A_3 \vdash_{\mathcal{A}} B_1 \leq B_3}$$

So the result $\Gamma \vdash \pi x{:}A_1.\, B_1 \leq \pi x{:}A_3.\, B_3$ follows using (AS-Π). The uses of the induction hypothesis are justified because in each, the maximal β_2+-reduction can be no longer than before, and the sum of the sizes of the terms is strictly smaller.

Case (AS-Λ): Similar to the π case.

Case (AS-APP-R): Consider the last rule in the derivation of $\Gamma \vdash_{\mathcal{A}} \ B \leq C$. If it is (AS-APP-R), then we get the result by transitivity of β_1-conversion using (AS-APP-R) again. Otherwise, the last rule must be (AS-APP-T). Lemma 3.10(2) and subject β_2-reduction (Proposition 3.2) imply:

$$\Gamma \vdash (\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2} : K \quad \text{and} \quad \Gamma \vdash (\Gamma(\alpha)\, M'_1 \cdots M'_n)^{\beta_2} : K.$$

Then we can apply the induction hypothesis and (AS-APP-T) to get this derivation:

$$\dfrac{\begin{array}{l}\Gamma \vdash_{\mathcal{A}} \ (\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2} \leq (\Gamma(\alpha)\, M'_1 \cdots M'_n)^{\beta_2} \\ \Gamma \vdash_{\mathcal{A}} \ (\Gamma(\alpha)\, M'_1 \cdots M'_n)^{\beta_2} \leq C\end{array}}{\dfrac{\Gamma \vdash_{\mathcal{A}} \ \Gamma(\alpha)\, M_1 \cdots M_n \leq C}{\Gamma \vdash_{\mathcal{A}} \ \alpha\, M_1 \cdots M_n \leq C}}$$

The first premise is an instance of reflexivity, since by Proposition 2.6(6), the two sides are β_1 convertible and we can use Proposition 3.5. Proposition 3.7 assures us that the new instance of transitivity has a strictly smaller measure because the sum of the lengths of the maximal β_2+-reductions in the new transitivity instance is strictly smaller.

Case (AS-APP-T): Lemma 3.10(2) and subject β_2-reduction (Proposition 3.2) imply:

$$\Gamma \vdash (\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2} : K.$$

We apply the induction hypothesis and (AS-APP-T) again to deduce:

$$\dfrac{\dfrac{\Gamma \vdash_{\mathcal{A}} \ (\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2} \leq B \qquad \Gamma \vdash_{\mathcal{A}} \ B \leq C}{\Gamma \vdash_{\mathcal{A}} \ (\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2} \leq C}}{\Gamma \vdash_{\mathcal{A}} \ \alpha\, M_1 \cdots M_n \leq C}$$

By Proposition 3.7 the new instance of transitivity has a strictly smaller measure. □

3.4 Completeness

Now we can establish completeness, using some properties of the new system.

Parts 2 and 3 of the next proposition hold for all M such that the normal forms mentioned exist (a weaker condition than kindability).

Proposition 3.12 (Properties of algorithmic subtyping).

1. *If* $\Gamma \vdash_{\mathcal{A}} \ A \leq B$ *and* $\Gamma =_{\beta_2} \Gamma'$, *then* $\Gamma' \vdash_{\mathcal{A}} \ A \leq B$.

2. If $\Gamma_1, x : A, \Gamma_2 \vdash_{\mathcal{A}} B \leq C$, *then* $\Gamma_1, \Gamma_2[x := M] \vdash_{\mathcal{A}} (B[x := M])^{\beta_2} \leq (C[x := M])^{\beta_2}$.

3. If $\Gamma \vdash_{\mathcal{A}} A \leq B$ *and* B *is not a* π*-type, then* $\Gamma \vdash_{\mathcal{A}} (AM)^{\beta_2} \leq (BM)^{\beta_2}$.
(Provided the normal forms in parts 2 and 3 exist).

Proof Parts 1 and 2 are proved by induction on derivations. Part 3 then follows by another induction on derivations, using parts 1 and 2. □

Part 3 of the proposition above is crucial in the completeness proof, where the induction hypothesis alone is too weak to show the admissibility of (S-APP).

Lemma 3.13 (Completeness of algorithmic subtyping).
If $\Gamma \vdash A \leq B$ *then* $\Gamma \vdash_{\mathcal{A}} A^{\beta_2} \leq B^{\beta_2}$.

Proof Using induction on the derivation of $\Gamma \vdash A \leq B$, considering the last rule:

Case (S-CONV): Using β_2-subject reduction for kinding, Proposition 3.2, $\Gamma \vdash A^{\beta_2}, B^{\beta_2} : K$. By Proposition 2.6(6) $A^{\beta_2} =_{\beta_1} B^{\beta_2}$ so we can use the admissibility of reflexivity, Proposition 3.5, to get the result.

Case (S-TRANS): By Proposition 3.11, since by Proposition 2.11 A, B, and C are kindable in the context.

Case (S-VAR): By Proposition 2.11, for some K, $\Gamma \vdash \alpha : K$ and $\Gamma \vdash \Gamma(\alpha) : K$. By subject reduction for kinding, $\Gamma \vdash \Gamma(\alpha)^{\beta_2} : K$ too. Then $\Gamma \vdash_{\mathcal{A}} \alpha \leq \Gamma(\alpha)^{\beta_2}$ via (AS-APP-T).

Case (S-π): Let the conclusion of the rule be $\Gamma \vdash \pi x{:}C.\, D \leq \pi x{:}C'.\, D'$. By the induction hypothesis and the premises, we get $\Gamma \vdash_{\mathcal{A}} C'^{\beta_2} \leq C^{\beta_2}$ and $\Gamma, x : C' \vdash_{\mathcal{A}} D^{\beta_2} \leq D'^{\beta_2}$. By Proposition 3.12(1), $\Gamma, x : C'^{\beta_2} \vdash_{\mathcal{A}} D^{\beta_2} \leq D'^{\beta_2}$ too. The result $\Gamma \vdash_{\mathcal{A}} (\pi x{:}C.\, D)^{\beta_2} \leq (\pi x{:}C'.\, D')^{\beta_2}$ follows using (AS-π).

Case (S-Λ): Similar to the previous case.

Case (S-APP): By the induction hypothesis and Proposition 3.12(3); the assumption that B is not a π-type follows from the premise $\Gamma \vdash BM : K$ and kinding generation. □

3.5 Equivalence

Combining the soundness and completeness lemmas, we get the following theorem.

Theorem 3.14 (Equivalence of algorithmic subtyping).
$\Gamma \vdash A \leq B$ *iff for some* K, $\Gamma \vdash A, B : K$ *and* $\Gamma \vdash_{\mathcal{A}} A^{\beta_2} \leq B^{\beta_2}$.

Proof By Lemma 3.13 with Proposition 2.11, and Lemma 3.4 using (CONV) and (TRANS). □

The equivalence of the two systems gives a powerful tool for analysing the subtyping relation. We can prove the generation principle we wanted.

Proposition 3.15 (Generation for subtyping).

1. *If* $\Gamma \vdash \alpha \leq C$ *and* α *is bounded in* Γ*, then either* $C =_\beta \alpha$*, or* $\Gamma \vdash \Gamma(\alpha) \leq C$*.*
2. *If* $\Gamma \vdash \pi x{:}A.\, B \leq C$ *then for some* A', B'*, (a)* $C =_\beta \pi x{:}A'.\, B'$*, (b)* $\Gamma \vdash A' \leq A$*, and (c)* $\Gamma, x : A' \vdash B \leq B'$*.*
3. *If* $\Gamma \vdash \Lambda x{:}A.\, B \leq C$ *then for some* B'*, (a)* $C =_\beta \Lambda x{:}A.\, B'$*, and (b)* $\Gamma, x : A \vdash B \leq B'$*.*

Proof Using Lemma 3.13, by considering the last rule of a derivation in the algorithmic system and then converting back to the original system using Lemma 3.4. □

There is no case for applications in this proposition. When $\Gamma \vdash A\,M \leq C$, we can only make deductions about the form of C based on $(A\,M)^{\beta_2}$; this reduces to one of the last two cases above, or a generalisation of the first (with $C =_{\beta_1} \alpha\,M_1 \cdots M_n$). To prove subject β_1-reduction, we only need part 2.

Proposition 3.16 (Closure under β_1-reduction).

1. *If* $\Gamma \vdash J$ *and* $\Gamma \longrightarrow\!\!\!\!\!\rightarrow_{\beta_1} \Gamma'$ *then* $\Gamma' \vdash J$*.*
2. *If* $\Gamma \vdash J$ *and* $J \longrightarrow\!\!\!\!\!\rightarrow_{\beta_1} J'$ *then* $\Gamma \vdash J'$*.*

Proof By simultaneous induction on derivations. The proof is similar to that of Proposition 3.2, except the case of an outermost reduction is in the rule (T-APP), where we use generation for both typing and subtyping. The one-step case of course relies on substitution, Proposition 2.8. □

3.6 Decidability

The "algorithmic" subtyping rules are syntax-directed, so they form an algorithm when viewed in reverse. Using the same measure used to prove the admissibility of transitivity, we can show that the algorithm for subtyping terminates on well-kinded types. Thus the subtyping relation is decidable for well-kinded types.

Lemma 3.17 (Decidability of subtyping for well-kinded types).
Let $\Gamma \vdash A : K_a$ *and* $\Gamma \vdash B : K_b$*. Then by applying the algorithmic subtyping rules, we can decide whether* $\Gamma \vdash_{\mathcal{A}} A^{\beta_2} \leq B^{\beta_2}$*. Moreover, this extends to a decision procedure for deciding* $\Gamma \vdash A \leq B$*.*

Proof For any algorithmic subtyping rule, $\mathsf{weight}_\Gamma(A_i, B_i)$ for each premise is strictly smaller than $\mathsf{weight}_\Gamma(A, B)$ for the conclusion. So every derivation ending in well-kinded types must be of finite height. The procedure of applying the rules backwards will either finish successfully (checking β_1 equalities in the leaves (AS-APP-R), which can be done by normalization), or else lead to a case where no rule is applicable. By Theorem 3.14 and the strong normalization property Proposition 2.3, this decides $\Gamma \vdash A \leq B$. □

Of course, we don't yet have an algorithm for determining the kinding relation so this lemma may not be useful — especially because to find whether types are well-kinded we have to do subtyping! In Section 4 we give an algorithm for kinding which only calls the subtyping algorithm on types which we already know to be well-kinded.

4 A type-checking algorithm

The next step towards proving decidability of $\lambda \mathrm{P}_{\leq}$ is to design algorithmic versions of the remaining judgements. In the same way that we removed kinding premises from subtyping, we remove formation premises from kinding and typing. Again this gives us something nearer a feasible algorithm, and helps prove termination.

Figures 2, 4 and 6 at the end of the paper show the new rules against the old ones, below we just give highlights. The new rules are syntax-directed, and with the convention that premises are evaluated in order (from left to right, "stacked" premises from top to bottom), they form a deterministic algorithm when viewed in reverse. Moreover, the rules for kinding and typing can be seen as *functions*, which given an input context and type (or term), yield a uniquely inferred kind (or type).

For formation, the rule for introducing a bounded type-variable becomes:

$$\frac{\Gamma \vdash_{\mathcal{A}} K \qquad \Gamma \vdash_{\mathcal{A}} A : K' \qquad K =_{\beta} K'}{\Gamma, \alpha \leq A : K \vdash_{\mathcal{A}} \star} \qquad \text{(AF-SUBTYPE)}$$

The first premise checks the well-formedness of the new kind K and the context. The second premise is used to find a kind K' for the bound A. Because the rules are syntax-directed, if K' exists, it is determined uniquely by A, so we can think of this as an inference procedure. Moreover, by the soundness property for the algorithmic system (Lemma 4.4), K' will be well-formed. This means that it is safe to check whether $K =_{\beta} K'$ by normalizing. Conversion is needed at this point because the conversion rule has been removed to make the system syntax-directed.

The algorithmic rule for kinding applications is:

$$\frac{\Gamma \vdash_{\mathcal{A}} A : \Pi x{:}B.K \qquad \Gamma \vdash_{\mathcal{A}} M : B' \qquad \Gamma \vdash_{\mathcal{A}} B'^{\beta_2} \leq B^{\beta_2}}{\Gamma \vdash_{\mathcal{A}} A\,M : K[x := M]} \qquad \text{(AK-APP)}$$

The first premise infers a unique kind for A, which must be a Π-kind if $A\,M$ is a valid application. As with the kinding rules, the algorithmic typing rules are syntax-directed, so the second premise infers a unique type B' for the argument M, if possible. Finally we must check that the inferred type B' is a subtype of the domain type B. Because the subtyping algorithm works on β_2-normal forms, we must normalize the two types before checking the subtyping relation. The normalization will be terminating because the kind and type inference procedure only infer valid types and kinds.

The third premise for (AK-APP) is necessary because subsumption is removed from the new typing relation. Similarly, we must allow subtyping when typing term applications:

$$\frac{\Gamma \vdash_{\mathcal{A}} M : A \qquad \mathsf{FLUB}_{\Gamma}(A) \equiv \pi x{:}B.C \qquad \Gamma \vdash_{\mathcal{A}} N : B' \qquad \Gamma \vdash_{\mathcal{A}} B'^{\beta_2} \leq B}{\Gamma \vdash_{\mathcal{A}} MN : C[x := N]} \qquad \text{(AT-APP)}$$

Once again, a subtyping check appears in the final premise. The inferred type of a function term need not have the form of a π-type; so in the second premise of the rule we invoke a function FLUB ("functional least upper bound") to search for a π-type for M. It climbs the context, following the subtyping order, until it finds a π-type or can go no farther. This is achieved by repeatedly β_2-normalizing and replacing head variables by their bounds.

$$\mathsf{FLUB}_\Gamma(A) = \begin{cases} \mathsf{FLUB}_{\Gamma|_\alpha}(\Gamma(\alpha)\, M_1 \cdots M_n) & A^{\beta_2} \equiv \alpha\, M_1 \cdots M_n \\ A^{\beta_2} & \text{otherwise} \end{cases}$$

(where the first case only applies if α is declared with a bound in Γ, and then $\Gamma|_\alpha$ is the initial prefix of Γ up to the declaration of α).

Now we must show that the algorithmic system is equivalent to the original one. First we prove some useful properties for the algorithm.

4.1 Basic properties of the algorithm

Proposition 4.1 (Context properties).

1. Renaming.
   Suppose θ is a mapping from variables to variables. Then $\Gamma \vdash_{\mathcal{A}} J$ implies $\theta(\Gamma) \vdash_{\mathcal{A}} \theta(J)$, where $\theta(-)$ denotes the obvious extensions of the mapping.

2. Thinning.
   Suppose $\Gamma \subseteq \Gamma'$, $\Gamma \vdash_{\mathcal{A}} J$ and $\Gamma' \vdash_{\mathcal{A}} \star$. Then $\Gamma' \vdash_{\mathcal{A}} J$.

Proposition 4.2 (Substitution in the algorithm).

1. *For subtyping. If $\Gamma_1, x : A, \Gamma_2 \vdash_{\mathcal{A}} C \leq D$, $\Gamma_1 \vdash_{\mathcal{A}} M : B$, and $\Gamma_1 \vdash B \leq A$ then $\Gamma_1, (\Gamma_2[x := M])^{\beta_2} \vdash_{\mathcal{A}} (C[x := M])^{\beta_2} \leq (D[x := M])^{\beta_2}$.*

2. *For the other judgements. If $\Gamma_1, x : A, \Gamma_2 \vdash_{\mathcal{A}} J$, $\Gamma_1 \vdash_{\mathcal{A}} M : B$, and $\Gamma_1 \vdash B \leq A$ then $\Gamma_1, \Gamma_2[x := M] \vdash_{\mathcal{A}} J[x := M]$.*

Proof By simultaneous induction on derivations, using Proposition 2.6 and Proposition 2.1. □

To prove equivalence, we also make use of some simple properties of FLUB, including the fact that $\mathsf{FLUB}_\Gamma(A)$ is an upper bound of A.

Proposition 4.3 (Properties of FLUB).

1. *If $\Gamma \vdash A : K$, then $\mathsf{FLUB}_\Gamma(A)$ is well-defined.*

2. *If $\mathsf{FLUB}_\Gamma(A)$ is defined and Γ is a prefix of the context Γ', then $\mathsf{FLUB}_{\Gamma'}(A) \equiv \mathsf{FLUB}_\Gamma(A)$.*

3. *If $\Gamma \vdash A : K$, then $\Gamma \vdash A \leq \mathsf{FLUB}_\Gamma(A)$.*

Proof

1. By assumption, A is normalizing and by Proposition 3.3(1), so too is $\Gamma(\alpha)\, M_1 \cdots M_n$ in the first case of the definition; the argument $\Gamma|_\alpha$ is shorter than Γ, which guarantees well-foundedness.

2. In the first case of the definition, $\mathsf{FLUB}_\Gamma(A) = \mathsf{FLUB}_{\Gamma|_\alpha}(\Gamma(\alpha)\, M_1 \cdots M_n)$ whilst $\mathsf{FLUB}_{\Gamma'}(A) = \mathsf{FLUB}_{\Gamma'|_\alpha}(\Gamma'(\alpha)\, M_1 \cdots M_n)$ but we must have $\alpha \in \mathit{Dom}(\Gamma)$, hence $\Gamma'(\alpha) = \Gamma(\alpha)$ and $\Gamma'|_\alpha = \Gamma|_\alpha$. The result is immediate for the second case.

3. By induction on the number of unfolding steps of $\mathsf{FLUB}_\Gamma(\ )$, a good measure since $\mathsf{FLUB}_\Gamma(A)$ is well-defined by part 1. In the base case, the second clause of the definition, $\mathsf{FLUB}_\Gamma(A) = A^{\beta_2}$ and the result follows by Proposition 3.2 and (S-CONV). In the step case, we know from Proposition 3.3(1) that $\Gamma \vdash \Gamma(\alpha)\, M_1 \cdots M_n : K$ and so we can use the induction hypothesis to obtain $\Gamma \vdash \Gamma(\alpha)\, M_1 \cdots M_n \leq \mathsf{FLUB}_\Gamma(\Gamma(\alpha)\, M_1 \cdots M_n)$. By part 2, $\mathsf{FLUB}_\Gamma(\Gamma(\alpha)\, M_1 \cdots M_n) \equiv \mathsf{FLUB}_\Gamma(\alpha\, M_1 \cdots M_n)$. The result follows using Proposition 3.3(2) and (S-TRANS). □

4.2 Equivalence

We prove that the algorithm is sound and complete for the original system defining $\lambda\mathrm{P}_{\leq}$. This is easier than it was for subtyping: the proofs proceed by induction on derivations in either system.

Lemma 4.4 (Soundness of algorithmic system).

1. $\Gamma \vdash_{\mathcal{A}} K$ *implies* $\Gamma \vdash K$.
2. *If* $\Gamma \vdash \star$ *then* $\Gamma \vdash_{\mathcal{A}} A : K$ *implies* $\Gamma \vdash A : K$.
3. *If* $\Gamma \vdash \star$ *then* $\Gamma \vdash_{\mathcal{A}} M : A$ *implies* $\Gamma \vdash M : A$.

Proof Simultaneously by induction on the derivation in the algorithmic system. Most cases follow immediately applying the corresponding rule of the original system, perhaps using the induction hypothesis. The case for (AF-SUBTYPE) uses Proposition 2.7(3). Here are the only two non-immediate cases:

Case (AT-APP): We have $\Gamma \vdash_{\mathcal{A}} M : A$, $\Gamma \vdash_{\mathcal{A}} N : B'$, $\Gamma \vdash_{\mathcal{A}} B'^{\beta_2} \leq B$ and $\mathsf{FLUB}_\Gamma(A) \equiv \pi x{:}B.\, C$. By the induction hypothesis for 3, $\Gamma \vdash M : A$ and by agreement $\Gamma \vdash A : \star$. By Proposition 4.3, $\Gamma \vdash A \leq \mathsf{FLUB}_\Gamma(A)$ and so by (T-SUB) $\Gamma \vdash M : \pi x{:}B.\, C$. Using the induction hypothesis for 3, $\Gamma \vdash N : B'$. Using Lemma 3.4 together with agreement, kinding generation and (S-CONV), we get $\Gamma \vdash B' \leq B$. Finally by (T-SUB), $\Gamma \vdash N : B$ and by (T-APP), $\Gamma \vdash MN : C[x := N]$.

Case (AK-APP): We have $\Gamma \vdash_{\mathcal{A}} A : \pi x{:}B.\, K$, $\Gamma \vdash_{\mathcal{A}} N : B'$ and $\Gamma \vdash_{\mathcal{A}} B'^{\beta_2} \leq B^{\beta_2}$. By the induction hypothesis for part 2, $\Gamma \vdash A : \pi x{:}B.\, K$ and by agreement, generation for kinding and context properties, $\Gamma \vdash B : \star$. By the induction hypothesis for part 3, $\Gamma \vdash M : B'$ and by agreement $\Gamma \vdash B' : \star$. Using Lemma 3.4, $\Gamma \vdash B'^{\beta_2} \leq B^{\beta_2}$ and by (S-CONV) and (S-TRANS), $\Gamma \vdash B' \leq B$. So using (T-SUB), $\Gamma \vdash M : B$ and by (K-APP), $\Gamma \vdash AM : K[x := M]$. □

For completeness we use the crucial characteristic of FLUB, which justifies its name: if a type A is a subtype of some π-type, then $\mathsf{FLUB}_\Gamma(A)$ is the least π-type greater than or equal to A in the subtype ordering. So an application typed with (AT-APP) is given a minimal type.

Proposition 4.5 (π-types and FLUB).

1. *If* $\Gamma \vdash A \leq \pi x{:}C.\, D$ *then* $\mathsf{FLUB}_\Gamma(A) \equiv \pi x{:}C'.\, D'$ *for some* C', D'.
2. *If* $\Gamma \vdash A \leq \pi x{:}C.\, D$ *then* $\Gamma \vdash \mathsf{FLUB}_\Gamma(A) \leq \pi x{:}C.\, D$.

Proof Each part via a corresponding result using induction in the algorithmic system.

1. We show by induction on the derivation that $\Gamma \vdash_{\mathcal{A}} A \leq \pi x{:}C.\, D$ implies $\mathsf{FLUB}_\Gamma(A) \equiv \pi x{:}C'.\, D'$.

   Case (AS-π): We are given that $A \equiv \pi x{:}C_1.\, D_1$ so the result follows by the definition of FLUB.

   Case (AS-APP-T): We are given that $A \equiv \alpha\, M_1 \cdots M_n$ and $\Gamma \vdash_{\mathcal{A}} (\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2} \leq \pi x{:}C_1.\, D_1$ By the induction hypothesis, $\mathsf{FLUB}_\Gamma((\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2}) \equiv \pi x{:}C_1'.\, D_1'$ for some C_1', D_1' and by the definition of FLUB and Proposition 4.3, it follows that $\mathsf{FLUB}_\Gamma(\alpha\, M_1 \cdots M_n) = \mathsf{FLUB}_\Gamma((\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2})$.

   Rules (AS-Λ) and (AS-APP-R) do not apply. The result for the original system follows from Lemma 3.13 and the properties of normal forms.

2. We show by induction on the derivation that $\Gamma \vdash_{\mathcal{A}} A \leq \pi x{:}C.\, D$ implies $\Gamma \vdash_{\mathcal{A}} \mathsf{FLUB}_\Gamma(A) \leq \pi x{:}C.\, D$.

   Case (AS-π): Then $A \equiv \mathsf{FLUB}_\Gamma(A) \equiv \pi x{:}C'.\, D'$, so the result is by assumption;

   Case (AS-APP-T): Then $A \equiv \alpha\, M_1 \cdots M_n$.
   By the induction hypothesis, $\Gamma \vdash_{\mathcal{A}} \mathsf{FLUB}_\Gamma((\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2}) \leq \pi x{:}C'.\, D'$.
   But by Proposition 4.3 and the definition of FLUB, it follows that $\mathsf{FLUB}_\Gamma(\alpha\, M_1 \cdots M_n) \equiv \mathsf{FLUB}_\Gamma((\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2})$ so we are done.

   Rules (AS-Λ) and (AS-APP-R) do not apply. The result for the original system follows using soundness and completeness of algorithmic subtyping, and a use of conversion and transitivity.

□

Lemma 4.6 (Completeness of algorithmic system).

1. *If* $\Gamma \vdash K$, *then* $\Gamma \vdash_{\mathcal{A}} K$.
2. *If* $\Gamma \vdash A : K$, *then there is a* K_a *such that* $\Gamma \vdash_{\mathcal{A}} A : K_a$, $K_a =_\beta K$, $\Gamma \vdash_{\mathcal{A}} K_a$, *and* $\Gamma \vdash_{\mathcal{A}} K$.
3. *If* $\Gamma \vdash M : A$, *then there is an* A_a *such that* $\Gamma \vdash_{\mathcal{A}} M : A_a$ *and* $\Gamma \vdash A_a \leq A$.

Proof Simultaneously by induction on derivations in the original system, using the corresponding algorithmic rules.

1. **Case** (F-EMPTY): By (AF-EMPTY).

   Case (F-TERM): Let the conclusion be $\Gamma_1, x : A \vdash \star$, and the premise be $\Gamma_1 \vdash A : \star$. By the induction hypothesis, there exists K_a, such that $\Gamma_1 \vdash_{\mathcal{A}} A : K_a$, $\Gamma_1 \vdash_{\mathcal{A}} K_a$, $\Gamma_1 \vdash_{\mathcal{A}} \star$, and $K_a =_\beta \star$. Since $\star$ can only be β-equal to itself, $K_a \equiv \star$. Then we get $\Gamma_1, x : A \vdash_{\mathcal{A}} \star$, by (AF-TERM).

   Case (F-TYPE): By the induction hypothesis and (AF-TYPE).

   Case (F-SUBTYPE): Let the premise be $\Gamma_1 \vdash A : K'$. By the induction hypothesis $\Gamma_1 \vdash_{\mathcal{A}} A : K'_a$, $\Gamma_1 \vdash_{\mathcal{A}} K'$ and $K'_a =_\beta K'$. We can now apply (AF-SUBTYPE).

   Case (F-π): By induction hypothesis and (AF-π). □

2. **Case** (K-VAR): By (AK-VAR), $K_a \equiv \mathsf{Kind}_\Gamma(\alpha)$. Let $\Gamma \equiv \Gamma_1, \alpha : \mathsf{Kind}_\Gamma(\alpha), \Gamma_2$. By generation of contexts (Proposition 2.7(2)) $\Gamma_1 \vdash \mathsf{Kind}_\Gamma(\alpha)$ with a shorter derivation. Then by the induction hypothesis, $\Gamma_1 \vdash_{\mathcal{A}} \mathsf{Kind}_\Gamma(\alpha)$. From $\Gamma \vdash \star$, by the induction hypothesis it follows that $\Gamma \vdash_{\mathcal{A}} \star$, and by Thinning (Proposition 4.1), $\Gamma \vdash_{\mathcal{A}} \mathsf{Kind}_\Gamma(\alpha)$.

   Case (K-π): Let the premise be $\Gamma, x : C \vdash B : \star$. By the induction hypothesis $\Gamma, x : C \vdash_{\mathcal{A}} B : K_b$ and $K_b =_\beta \star$. Since the only kind β-equal to $\star$ is itself, $K_b \equiv \star$.

   By generation of contexts (Proposition 2.7(2)), $\Gamma \vdash C : \star$ with a shorter derivation. Hence by the induction hypothesis, $\Gamma \vdash_{\mathcal{A}} C : \star$, and by (AK-$\pi$), $\Gamma \vdash_{\mathcal{A}} \pi x{:}C.\, B : \star$.

   By well-formedness of contexts (Proposition 2.7(3)), from $\Gamma, x : C \vdash B : \star$ it follows that $\Gamma \vdash \star$ with a shorter derivation, hence by the induction hypothesis, $\Gamma \vdash_{\mathcal{A}} \star$.

   Case (K-Λ): Let the premise be $\Gamma, x : C \vdash B : K$. By the induction hypothesis, there exists K_b such that $\Gamma, x : C \vdash_{\mathcal{A}} B : K_b$, $K_b =_\beta K$, $\Gamma, x : C \vdash_{\mathcal{A}} K_b$, and $\Gamma, x : C \vdash_{\mathcal{A}} K$. By (AF-$\pi$), $\Gamma \vdash_{\mathcal{A}} \Pi x{:}C.K$ and $\Gamma \vdash_{\mathcal{A}} \Pi x{:}C.K_b$, and by the definition of β-equality $\Pi x{:}C.K_b =_\beta \Pi x{:}C.K$. From the premise, by generation of contexts (Proposition 2.7(2)), $\Gamma \vdash C : \star$ with a shorter derivation, and by the induction hypothesis, $\Gamma \vdash_{\mathcal{A}} C : \star$. We can now apply (AK-$\Lambda$) to obtain $\Gamma \vdash_{\mathcal{A}} \Lambda x{:}C.\, B : \Pi x{:}C.K_b$. Take $K_a \equiv \Pi x{:}C.K_b$.

   Case (K-APP): Let the premises be $\Gamma \vdash C : \Pi x{:}B.K$ and $\Gamma \vdash M : B$. By the induction hypothesis on the second premise there exists B_b such that $\Gamma \vdash_{\mathcal{A}} M : B_b$ and $\Gamma \vdash B_b \leq B$. By the induction hypothesis on the first premise, there exists K_c such that $\Gamma \vdash_{\mathcal{A}} C : K_c$, $\Gamma \vdash_{\mathcal{A}} K_c$, $\Gamma \vdash_{\mathcal{A}} \Pi x{:}B.K$, and $K_c =_\beta \Pi x{:}B.K$. By Church Rosser, $K_c \equiv \Pi x{:}B.'K'$ with $B' =_\beta B$ and $K' =_\beta K$. By inversion of (AF-π), $\Gamma \vdash_{\mathcal{A}} \Pi x{:}B.'K'$ implies $\Gamma, x : B' \vdash_{\mathcal{A}} K'$. By soundness, $\Gamma, x : B' \vdash K'$ and by generation of contexts (Proposition 2.7), $\Gamma \vdash B' : \star$.

   By agreement (Proposition 2.11), $\Gamma \vdash C : \Pi x{:}B.K$ implies $\Gamma \vdash \Pi x{:}B.K$, and from $\Gamma \vdash M : B$ it follows that $\Gamma \vdash B : \star$.

   From, $B' =_\beta B$, $\Gamma \vdash B : \star$ and $\Gamma \vdash B' : \star$, by (S-CONV), $\Gamma \vdash B \leq B'$. By (S-TRANS), $\Gamma \vdash B_b \leq B'$, and by equivalence (Theorem 3.14), $\Gamma \vdash_{\mathcal{A}} {B_b}^{\beta_2} \leq {B'}^{\beta_2}$. Hence, by (AK-APP), $\Gamma \vdash_{\mathcal{A}} CM : K'[x := M]$.

From $\Gamma, x : B' \vdash_{\mathcal{A}} K'$, $\Gamma \vdash_{\mathcal{A}} M : B_b$ and $\Gamma \vdash B_b \leq B'$, by substitution in the algorithm (Proposition 4.2), $\Gamma \vdash_{\mathcal{A}} K'[x := M]$. By inversion of (AF-$\pi$), $\Gamma \vdash_{\mathcal{A}} \Pi x{:}B.K$ implies $\Gamma, x : B \vdash_{\mathcal{A}} K$, and since $\Gamma \vdash B_b \leq B$, again by substitution (Proposition 4.2), $\Gamma \vdash_{\mathcal{A}} K[x := M]$. Finally, by Proposition 2.6(4), $K'[x := M] =_\beta K[x := M]$. Take $K_a \equiv K'[x := M]$.

Case (K-CONV): By induction hypothesis and transitivity of $=_\beta$.

3. **Case** (T-VAR): By (AT-VAR). We get $\Gamma \vdash \Gamma(x) \leq \Gamma(x)$ by (S-CONV) and agreement (Proposition 2.11).

   Case (T-λ): Let the premise be $\Gamma, x : C \vdash M : B$. By the induction hypothesis, there exists B_b such that, $\Gamma, x : C \vdash_{\mathcal{A}} M : B_b$ and $\Gamma, x : C \vdash B_b \leq B$. By generation of contexts, there is a shorter derivation of $\Gamma \vdash C : \star$ and by the induction hypothesis $\Gamma \vdash_{\mathcal{A}} C : \star$, since there exists K_c with $K_c =_\beta \star$, and hence $K_c \equiv \star$. Then by (AT-λ) $\Gamma \vdash_{\mathcal{A}} \lambda x{:}C.\, M : \pi x{:}C.\, B_b$.

   From $\Gamma \vdash C : \star$, by (S-CONV), $\Gamma \vdash C \leq C$. From $\Gamma, x : C \vdash B_b \leq B$, by agreement, uniqueness of kinds, and (K-π), $\Gamma \vdash \pi x{:}C.\, B : \star$. Finally, by (S-$\pi$), $\Gamma \vdash \pi x{:}C.\, B_b \leq \pi x{:}C.\, B$. Take $A_a \equiv \pi x{:}C.\, B_b$.

   Case (T-APP): Suppose the premises are $\Gamma \vdash N : \pi x{:}B.\, C$ and $\Gamma \vdash N' : B$. By induction hypothesis, $\Gamma \vdash_{\mathcal{A}} N : D_a$ with $\Gamma \vdash D_a \leq \pi x{:}B.\, C$ and $\Gamma \vdash_{\mathcal{A}} N' : B_a$ with $\Gamma \vdash B_a \leq B$. Hence by Proposition 4.5, $\mathsf{FLUB}_\Gamma(D_a) \equiv \pi x{:}B'.\, C'$ and $\Gamma \vdash \pi x{:}B'.\, C' \leq \pi x{:}B.\, C$. By subtyping generation, $\Gamma \vdash B \leq B'$ and by (S-TRANS) $\Gamma \vdash B_a \leq B'$. By the completeness of algorithmic subtyping, $\Gamma \vdash_{\mathcal{A}} {B_a}^{\beta_2} \leq {B'}^{\beta_2}$. Observe that since $\mathsf{FLUB}_\Gamma(D_a)$ is in β_2-normal form, ${B'}^{\beta_2} \equiv B'$. So we can apply (AT-APP) to get $\Gamma \vdash_{\mathcal{A}} N\, N' : A_a$ where $A_a \equiv C'[x := N']$. Finally, $\Gamma \vdash A_a \leq C[x := N']$ follows from subtyping generation and substitution for the subtyping judgement.

   Case (T-SUB): By induction hypothesis and (S-TRANS).

We now have an equivalence between the two systems.

Theorem 4.7 (Equivalence).

1. $\Gamma \vdash A \leq B$ *iff for some* K, $\Gamma \vdash A, B : K$ *and* $\Gamma \vdash_{\mathcal{A}} A^{\beta_2} \leq B^{\beta_2}$.
2. $\Gamma \vdash A : K$ *iff for some* K_a, $\Gamma \vdash_{\mathcal{A}} A : K_a$, $K_a =_\beta K$ *and* $\Gamma \vdash_{\mathcal{A}} K$.
3. $\Gamma \vdash M : A$ *iff for some* A_a, $\Gamma \vdash_{\mathcal{A}} M : A_a$ *and* $\Gamma \vdash A_a \leq A$.
4. $\Gamma \vdash K$ *iff* $\Gamma \vdash_{\mathcal{A}} K$.

Proof For part 1, the equivalence for subtyping was proved in Theorem 3.14. The right-to-left direction of parts 2–4 follow by soundness, Lemma 4.4, using (T-SUB) for part 2 and Proposition 2.7(3) and (K-CONV) for part 3. The left-to-right directions follow by completeness, Lemma 4.6. □

This theorem also proves that the algorithmic typing rules assign a minimal type to a typable term. The minimal typing property for the original system follows, by soundness.

Corollary 4.8 (Minimal typing property for $\lambda P_{\leq}$).
Suppose M is typable in Γ. Then there is an A_a such that

1. $\Gamma \vdash M : A_a$ *and*
2. *whenever* $\Gamma \vdash M : A$, *then* $\Gamma \vdash A_a \leq A$.

Proof By Lemma 4.6, there is an A_a such that $\Gamma \vdash_{\mathcal{A}} M : A_a$. By Lemma 4.4, $\Gamma \vdash M : A_a$ too, showing part 1. Because the algorithmic system is deterministic, A_a only depends on Γ and M; it is necessarily unique. So for *any* A such that $\Gamma \vdash M : A$, we have $\Gamma \vdash A_a \leq A$ by Lemma 4.6, showing part 2. □

4.3 Decidability

Our final theorem establishes the decidability of the algorithmic judgements, which guarantees the termination of subtype checking, kind inference, minimal type inference and formation checking.

Theorem 4.9 (Decidability of algorithmic judgements).

1. *If* $\Gamma \vdash A : K_a$ *and* $\Gamma \vdash B : K_b$, *then it is decidable whether* $\Gamma \vdash_{\mathcal{A}} A^{\beta_2} \leq B^{\beta_2}$.
2. *If* $\Gamma \vdash \star$, *it is decidable whether there exists a* K_a *such that* $\Gamma \vdash_{\mathcal{A}} A : K_a$.
3. *If* $\Gamma \vdash \star$, *it is decidable whether there exists an* A_a *such that* $\Gamma \vdash_{\mathcal{A}} M : A_a$.
4. *It is decidable whether* $\Gamma \vdash_{\mathcal{A}} K$.

Proof Each case follows by induction on a measure that decreases from conclusion to premises in each rule. Part 1 was proved in Lemma 3.17. Parts 2 and 3 using part 1 by simultaneous induction on the size of the subject (the term to the left of ":"), and part 4 by induction on the size of the judgement, using parts 1–3.

2. The size of the subject decreases from conclusion to kinding and typing premises. It remains to prove that checking the subtyping premise in (AK-APP) terminates. For that it is enough to show that $\Gamma \vdash B'^{\beta_2} : \star$ and $\Gamma \vdash B^{\beta_2} : \star$, because, by part 1, the call to the subtyping algorithm $\Gamma \vdash_{\mathcal{A}} B'^{\beta_2} \leq B^{\beta_2}$ terminates.

   By soundness (Lemma 4.4), $\Gamma \vdash M : B'$, and by agreement (Proposition 2.11) $\Gamma \vdash B' : \star$. By strong normalization (Proposition 2.3), B'^{β_2} exists, and by β_2-subject reduction (Proposition 3.2), $\Gamma \vdash B'^{\beta_2} : \star$.

   By soundness and agreement, $\Gamma \vdash \Pi x{:}B.K$, and because of the determinacy of the formation rules $\Gamma, x : B \vdash K$. By generation of contexts (Proposition 2.7(2)), $\Gamma \vdash B : \star$, then by strong normalization and β_2-subject reduction, $\Gamma \vdash B^{\beta_2} : \star$.

3. The rule (AT-λ) is the only rule that has a kinding premise, and is why we need to consider kinding and typing together. The size of the subject of each typing or kinding premise is strictly smaller than the size of the subject of the conclusion, so there can be no infinite paths in the typing or kinding rules. The interesting case is (AT-APP). By soundness we know that $\Gamma \vdash M : A$ after the first premise; by agreement this guarantees $\Gamma \vdash A : \star$, so by Proposition 4.3(1) and its proof, $\mathsf{FLUB}_\Gamma(A)$ is defined and the process of calculating it terminates. If we can establish that $\Gamma \vdash B'^{\beta_2} : \star$ and $\Gamma \vdash B : \star$ then, by part 1, the call to the subtyping algorithm $\Gamma \vdash_{\mathcal{A}} B'^{\beta_2} \leq B$ terminates. (Notice by the definition of FLUB that B is already in β_2-normal form.)

   By soundness and agreement, $\Gamma \vdash B' : \star$ and by strong normalization and β_2-subject reduction $\Gamma \vdash B'^{\beta_2} : \star$.

   By Proposition 4.3(3), agreement and uniqueness of kinds (Proposition 2.9), $\Gamma \vdash \pi x{:}B.\, C : \star$, by generation for kinding, $\Gamma, x : B \vdash C : \star$, and by generation for contexts $\Gamma \vdash B : \star$.

4. In each formation premise the total number of symbols in the judgement is strictly smaller than in the conclusion, so there can be no infinite path of formation rules. In (AF-TERM) and (AF-SUBTYPE), by Lemma 4.4 and the first premise, we have $\Gamma \vdash \star$ and so by part 2 the algorithm for checking or inferring kinds will terminate. □

Theorem 4.9, together with the equivalence proved in Theorem 4.7, shows that we have a correct and terminating algorithm for deciding any judgement of the original presentation. Here is the argument.

The formation judgement is primary; $\Gamma \vdash K$ holds iff $\Gamma \vdash_{\mathcal{A}} K$, and $\Gamma \vdash_{\mathcal{A}} K$ is decidable. For the other judgements, we must be careful to invoke the algorithmic judgements only when we know the context and pre-terms to be well-formed.

Recall that the rules for kind and type checking allow no conversion or subsumption for an arbitrary kind or type, so viewed in reverse they form deterministic functions, for kind and type *inference*. So given a type, we can compute a kind of it, provided one exists. Given a term, we can compute a type of it. Again, this is subject to checking well-formedness of the context first.

Consider the subtyping judgement. To check if $\Gamma \vdash A \leq B$, we use the following four steps:

$$\frac{\begin{array}{ll}1. & \Gamma \vdash_{\mathcal{A}} \star \\ 2. & \Gamma \vdash_{\mathcal{A}} A : K_a \\ 3. & \Gamma \vdash_{\mathcal{A}} B : K_b \\ 4. & \Gamma \vdash_{\mathcal{A}} A^{\beta_2} \leq B^{\beta_2}\end{array}}{\Gamma \vdash A \leq B} \quad \text{(SUBTYPE-CHECK)}$$

The first step checks whether Γ is a well-formed context, which is decidable, and by soundness $\Gamma \vdash \star$. Knowing this, we see whether A and B have kinds, say K_a and K_b. These are synthesized by the algorithmic rules for kind-checking. Furthermore, kind-checking in a well-formed context is decidable. If kinds can be found for A and B, by soundness we

know that $\Gamma \vdash A : K_a$ and $\Gamma \vdash B : K_b$, which implies that step 4 is decidable. Finally, by equivalence, if any step fails, then $\Gamma \vdash A \leq B$ does not hold.

The typing and kinding judgements yield similar procedures.

Consider now the kinding judgement. To check if $\Gamma \vdash A : K$, we use the following four steps:

$$\frac{\begin{array}{ll}1. & \Gamma \vdash_{\mathcal{A}} K \\ 2. & \Gamma \vdash_{\mathcal{A}} A : K_a \\ 3. & \Gamma \vdash_{\mathcal{A}} K_a \\ 4. & K_a =_\beta K\end{array}}{\Gamma \vdash A : K} \quad \text{(KIND-CHECK)}$$

Step 1 checks that the given kind K is well-formed in the context Γ, and it is decidable. By soundness, we have that $\Gamma \vdash K$, and by well-formedness of contexts Proposition 2.7(3), $\Gamma \vdash \star$. Hence it is decidable whether there exists K_a such that $\Gamma \vdash_{\mathcal{A}} A : K_a$, which is step 2. If there is such a K_a, we check in step 3 if $\Gamma \vdash_{\mathcal{A}} K_a$, which is also decidable. If so, by soundness, $\Gamma \vdash K_a$, and by strong normalization (Proposition 2.3) step 4 is decidable, because we can check if $K_a =_\beta K$ by reducing both K and K_a to normal form and compare them. Finally, by equivalence, if any step fails, then $\Gamma \vdash A : K$ does not hold.

Finally, let us consider the typing judgement. To check whether $\Gamma \vdash M : A$, we use the following five steps:

$$\frac{\begin{array}{ll}1. & \Gamma \vdash_{\mathcal{A}} \star \\ 2. & \Gamma \vdash_{\mathcal{A}} M : A_a \\ 3. & \Gamma \vdash_{\mathcal{A}} A_a : K_a \\ 4. & \Gamma \vdash_{\mathcal{A}} A : K \\ 5. & \Gamma \vdash_{\mathcal{A}} {A_a}^{\beta_2} \leq A^{\beta_2}\end{array}}{\Gamma \vdash M : A} \quad \text{(TYPE-CHECK)}$$

Step 1 is decidable, and by soundness, $\Gamma \vdash \star$. The second step infers a type A_a for M in Γ, which is decidable because $\Gamma \vdash \star$. In steps 3 and 4, kinds K_a for A_a and K for A are inferred. We know that the last two steps are decidable, because $\Gamma \vdash \star$. If such kinds exist, it finally ckecks wether $\Gamma \vdash_{\mathcal{A}} {A_a}^{\beta_2} \leq A^{\beta_2}$, which is decidable for well-kinded types A_a and A. Finally, by equivalence, if any step fails, then $\Gamma \vdash M : A$ does not hold.

As stated, these procedures are of theoretical interest only; we expect that practical implementations would make use of β_2-weak-head normal forms instead of full β_2-normal forms, amongst other efficiency improvements.

Finally we can state our main result.

Corollary 4.10 (Decidability). *Each judgement of $\lambda\mathrm{P}_{\leq}$ is decidable.*

5 Conclusion

Our system $\lambda\mathrm{P}_{\leq}$ adds subtyping to $\lambda\mathrm{P}$. The system $\lambda\mathrm{P}$ is the simplest corner of Barendregt's λ-cube with type dependency, yet it is the core of many applied type theories for which subtyping is desirable. Subtyping posed a challenge for meta-theoretical study;

we met the challenge by proving properties in a carefully chosen order and formulating an algorithmic version of the system. The main result is the decidability of the typing and subtyping relations, achieved using non-trivial extensions of work that dates back to Cardelli's early ideas [8], Curien and Ghelli's analysis of $F_{\leq}$ [16] and subsequent studies of non-dependent subtyping systems [22, 13, 26].

Of the related work when we began, Pfenning's study of refinement types [21] is closest. There, a *sort* is declared as a *refinement* of a type, and there is a *subsorting* relation. Whilst subsorting is a richer relation than our subtyping (for example, intersections of sorts are permitted), there is a strict separation between types and sorts to ensure a straightforward proof of decidability of the system. Sorts cannot appear in labels of λ-abstractions, so it is impossible to write functions with domains limited via subsorting, a disadvantage Pfenning mentions. No such restriction applies to our calculus, where subtyping applies uniformly.

Other early related work includes that of Cardelli [8, 9], who gave basic definitions and ideas about semi-decision procedures; Aspinall [1], who describes a system that has subtyping and dependent types but no type variables; Coquand [14] who considers subtyping inductive data types in a dependent type theory, and Betarte and Tasistro who investigated adding dependent records to Martin-Löf's type theory [6].

There are several ways to continue the work begun here. One goal is to find a semantics for $\lambda\mathrm{P}_{\leq}$. The ideal would be to translate $\lambda\mathrm{P}_{\leq}$ into $\lambda\mathrm{P}$ by removing subtyping, along the lines of [7]. We hinted at this understanding in Section 1 when we suggested that the injection function "!" is implicit in the presence of subtyping, as if inserted automatically. To generalise, we must assume families of coercions for each bounded type variable in a $\lambda\mathrm{P}_{\leq}$ context, and show that there is a canonical way of inserting coercions to translate pre-terms at each level to $\lambda\mathrm{P}$. Then any model of $\lambda\mathrm{P}$ will serve as a model of $\lambda\mathrm{P}_{\leq}$ and the class of logics that can be encoded will be the same as for LF.

For the application of logic encoding, it is well known that including η-conversion in the framework is important. Studying examples, the need for intersection types which Pfenning recognised also seems important, allowing constants to be overloaded. If the techniques of [13] can be adapted, we could reproduce Pfenning's examples in [21].

In another direction, we need to examine richer type systems, adding the polymorphism and bounded quantification of $F_{\leq}$, and approaching the type theories underlying the proof assistants mentioned in the introduction. We suspect that a careful combination of these features would also give a good type system for a programming language, although investigation of programming with type-dependency alone is in its infancy. And to integrate our work into real proof assistants, we must consider more than type-checking, since systems like *Elf* and *LEGO* do more than check proofs. Searching for a proof or applying a tactic involves unification and matching procedures which would need modification to take subtyping into account.

Finally, it would be nice to lift the results to a more general setting, pursuing the idea of adding subtyping to the Calculus of Constructions [15] or to Pure Type Systems [5]. It is easy to formulate such extensions, maybe using Cardelli's power types [9], but it seems much harder to prove things about them. We believe that variations of the techniques used here may help. Indeed, since publication of [3], this has been achieved. Extending the Calculus of Construction with subtyping has been undertaken by Chen, starting from a system similar

to our algorithm [10], and Zwanenburg has extended Pure Type Systems [27]. Both avoid circularity problems by defining subtyping on pre-terms in the first place; but then one is obliged to show that the resulting relation is the intended one on well-formed types, which amounts to proving equivalence results broadly similar to ours. Compagnoni and Goguen have used another technique, Typed Operational Semantics, to study a higher order calculus with bounded operator abstraction and subtyping, containing similar circularities to the ones here [12, 11]. In other related work, Luo has developed a system of *coercive subtyping* intended for dependent type theories of proof assistants, where a subtyping relation is induced by coercion functions [19].

Acknowledgements. Our sincere thanks are due to R. Constable, M. Dezani, H. Goguen, M. Hofmann, Z. Luo, R. Pollack and D. Sannella for their comments on drafts. The TCS referees gave us useful suggestions for clarifications.

We gratefully acknowledge the support provided by the LFCS, a UK EPSRC studentship and EPSRC grant GR/H73103 (DA), and EPSRC grants GR/K38403, GR/G55792, and CONFER and EuroFOCS (AC).

$$\frac{}{\langle\rangle \vdash \star} \quad (\text{F-EMPTY})$$

$$\frac{\Gamma \vdash A : \star}{\Gamma, x : A \vdash \star} \quad (\text{F-TERM})$$

$$\frac{\Gamma \vdash K}{\Gamma, \alpha : K \vdash \star} \quad (\text{F-TYPE})$$

$$\frac{\Gamma \vdash A : K}{\Gamma, \alpha \leq A : K \vdash \star} \quad (\text{F-SUBTYPE})$$

$$\frac{\Gamma, x : A \vdash K}{\Gamma \vdash \Pi x{:}A.K} \quad (\text{F-}\Pi)$$

Figure 1: Formation of contexts and kinds

$$\frac{}{\langle\rangle \vdash_{\mathcal{A}} \star} \quad (\text{AF-EMPTY})$$

$$\frac{\Gamma \vdash_{\mathcal{A}} \star \qquad \Gamma \vdash_{\mathcal{A}} A : \star}{\Gamma, x : A \vdash_{\mathcal{A}} \star} \quad (\text{AF-TERM})$$

$$\frac{\Gamma \vdash_{\mathcal{A}} K}{\Gamma, \alpha : K \vdash_{\mathcal{A}} \star} \quad (\text{AF-TYPE})$$

$$\frac{\Gamma \vdash_{\mathcal{A}} K \quad \Gamma \vdash_{\mathcal{A}} A : K' \qquad K =_\beta K'}{\Gamma, \alpha \leq A : K \vdash_{\mathcal{A}} \star} \quad (\text{AF-SUBTYPE})$$

$$\frac{\Gamma, x : A \vdash_{\mathcal{A}} K}{\Gamma \vdash_{\mathcal{A}} \Pi x{:}A.K} \quad (\text{AF-}\Pi)$$

Figure 2: Algorithmic Formation

$$\frac{\Gamma \vdash \star \qquad \alpha \in Dom(\Gamma)}{\Gamma \vdash \alpha : \mathsf{Kind}_\Gamma(\alpha)} \quad (\text{K-VAR})$$

$$\frac{\Gamma, x : A \vdash B : \star}{\Gamma \vdash \pi x{:}A.\, B : \star} \quad (\text{K-}\pi)$$

$$\frac{\Gamma, x : A \vdash B : K}{\Gamma \vdash \Lambda x{:}A.\, B : \Pi x{:}A.K} \quad (\text{K-}\Lambda)$$

$$\frac{\Gamma \vdash A : \Pi x{:}B.K \qquad \Gamma \vdash M : B}{\Gamma \vdash A\, M : K[x := M]} \quad (\text{K-APP})$$

$$\frac{\Gamma \vdash A : K \qquad \Gamma \vdash K' \qquad K =_\beta K'}{\Gamma \vdash A : K'} \quad (\text{K-CONV})$$

Figure 3: Kinding

$$\frac{\alpha \in Dom(\Gamma)}{\Gamma \vdash_{\mathcal{A}} \alpha : \mathsf{Kind}_\Gamma(\alpha)} \quad (\text{AK-VAR})$$

$$\frac{\Gamma \vdash_{\mathcal{A}} A : \star \qquad \Gamma, x : A \vdash_{\mathcal{A}} B : \star}{\Gamma \vdash_{\mathcal{A}} \pi x{:}A.\, B : \star} \quad (\text{AK-}\pi)$$

$$\frac{\Gamma \vdash_{\mathcal{A}} A : \star \qquad \Gamma, x : A \vdash_{\mathcal{A}} B : K}{\Gamma \vdash_{\mathcal{A}} \Lambda x{:}A.\, B : \Pi x{:}A.K} \quad (\text{AK-}\Lambda)$$

$$\frac{\Gamma \vdash_{\mathcal{A}} A : \Pi x{:}B.K \qquad \begin{array}{l}\Gamma \vdash_{\mathcal{A}} M : B' \\ \Gamma \vdash_{\mathcal{A}} B'^{\beta_2} \leq B^{\beta_2}\end{array}}{\Gamma \vdash_{\mathcal{A}} A\, M : K[x := M]} \quad (\text{AK-APP})$$

Figure 4: Algorithmic Kinding

$$\frac{\Gamma \vdash \star \qquad x \in Dom(\Gamma)}{\Gamma \vdash x : \Gamma(x)} \quad (\text{T-VAR})$$

$$\frac{\Gamma, x : A \vdash M : B}{\Gamma \vdash \lambda x{:}A.\, M : \pi x{:}A.\, B} \quad (\text{T-}\lambda)$$

$$\frac{\Gamma \vdash M : \pi x{:}A.\, B \qquad \Gamma \vdash N : A}{\Gamma \vdash MN : B[x := N]} \quad (\text{T-APP})$$

$$\frac{\Gamma \vdash M : A \qquad \Gamma \vdash A \leq B}{\Gamma \vdash M : B} \quad (\text{T-SUB})$$

Figure 5: Typing

$$\frac{x \in Dom(\Gamma)}{\Gamma \vdash_{\mathcal{A}} x : \Gamma(x)} \quad (\text{AT-VAR})$$

$$\frac{\Gamma \vdash_{\mathcal{A}} A : \star \qquad \Gamma, x : A \vdash_{\mathcal{A}} M : B}{\Gamma \vdash_{\mathcal{A}} \lambda x{:}A.\, M : \pi x{:}A.\, B} \quad (\text{AT-}\lambda)$$

$$\frac{\begin{array}{ll}\Gamma \vdash_{\mathcal{A}} M : A & \Gamma \vdash_{\mathcal{A}} N : B' \\ \mathsf{FLUB}_\Gamma(A) \equiv \pi x{:}B.\, C & \Gamma \vdash_{\mathcal{A}} {B'}^{\beta_2} \leq B\end{array}}{\Gamma \vdash_{\mathcal{A}} MN : C[x := N]} \quad (\text{AT-APP})$$

Figure 6: Algorithmic Typing

$$\frac{\Gamma \vdash A : K \qquad \Gamma \vdash B : K \qquad A =_\beta B}{\Gamma \vdash A \leq B} \quad (\text{S-CONV})$$

$$\frac{\Gamma \vdash A \leq B \qquad \Gamma \vdash B \leq C}{\Gamma \vdash A \leq C} \quad (\text{S-TRANS})$$

$$\frac{\Gamma \vdash \star \qquad \alpha \text{ bounded in } \Gamma}{\Gamma \vdash \alpha \leq \Gamma(\alpha)} \quad (\text{S-VAR})$$

$$\frac{\begin{array}{l}\Gamma \vdash A' \leq A \\ \Gamma, x : A' \vdash B \leq B' \qquad \Gamma \vdash \pi x{:}A.\, B : \star\end{array}}{\Gamma \vdash \pi x{:}A.\, B \leq \pi x{:}A'.\, B'} \quad (\text{S-}\pi)$$

$$\frac{\Gamma, x : A \vdash B \leq B'}{\Gamma \vdash \Lambda x{:}A.\, B \leq \Lambda x{:}A.\, B'} \quad (\text{S-}\Lambda)$$

$$\frac{\Gamma \vdash A \leq B \qquad \Gamma \vdash BM : K}{\Gamma \vdash A\, M \leq B\, M} \quad (\text{S-APP})$$

Figure 7: Subtyping

$$\frac{\Gamma \vdash_{\mathcal{A}} A' \leq A \qquad \Gamma, x : A' \vdash_{\mathcal{A}} B \leq B'}{\Gamma \vdash_{\mathcal{A}} \pi x{:}A.\, B \leq \pi x{:}A'.\, B'} \quad (\text{AS-}\pi)$$

$$\frac{A =_{\beta_1} A' \qquad \Gamma, x : A' \vdash_{\mathcal{A}} B \leq B'}{\Gamma \vdash_{\mathcal{A}} \Lambda x{:}A.\, B \leq \Lambda x{:}A'.\, B'} \quad (\text{AS-}\Lambda)$$

$$\frac{M_1 =_{\beta_1} M_1' \;\cdots\; M_n =_{\beta_1} M_n'}{\Gamma \vdash_{\mathcal{A}} \alpha\, M_1 \cdots M_n \;\leq\; \alpha\, M_1' \cdots M_n'} \quad (\text{AS-APP-R})$$

$$\frac{\begin{array}{l}\alpha \text{ bounded in } \Gamma,\ A \not\equiv \alpha\, M_1 \cdots M_n \\ \Gamma \vdash_{\mathcal{A}} (\Gamma(\alpha)\, M_1 \cdots M_n)^{\beta_2} \leq A\end{array}}{\Gamma \vdash_{\mathcal{A}} \alpha\, M_1 \cdots M_n \leq A} \quad (\text{AS-APP-T})$$

Figure 8: Algorithmic Subtyping

References

[1] David Aspinall. Subtyping with singleton types. In *Proc. Computer Science Logic, CSL'94, Kazimierz, Poland*, Lecture Notes in Computer Science 933. Springer-Verlag, 1995.

[2] David Aspinall. *Type Systems for Modular Programs and Specification*. PhD thesis, Department of Computer Science, University of Edinburgh, 1997.

[3] David Aspinall and Adriana Compagnoni. Subtyping dependent types. In E. Clarke, editor, *Proceedings, Eleventh Annual IEEE Symposium on Logic in Computer Science*, pages 86–97, New Brunswick, New Jersey, 1996. IEEE Computer Society Press.

[4] Arnon Avron, Furio Honsell, Ian A. Mason, and Robert Pollack. Using typed lambda calculus to implement formal systems on a machine. *Journal of Automated Reasoning*, 9:309–354, 1992.

[5] Henk Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science*. Oxford University Press, 1992.

[6] Gustavo Betarte and Alvaro Tasistro. Extension of Martin-Löf's type theory with record types and subtyping. In *Proceedings of 25 Years of Constructive Type Theory*. Oxford University Press, 1997.

[7] Val Breazu-Tannen, Thierry Coquand, Carl A. Gunter, and Andre Scedrov. Inheritance as implicit coercion. *Information and Computation*, 93:172–221, 1991.

[8] Luca Cardelli. Typechecking dependent types and subtypes. In M. Boscarol, L. Carluccia Aiello, and G. Levi, editors, *Proc. of the Workshop on Foundations of Logic and Functional Programming*, Lecture Notes in Computer Science 306. Springer-Verlag, 1987.

[9] Luca Cardelli. Structural subtyping and the notion of power type. In *Conference Record of the Fifteenth Annual ACM Symposium on Principles of Programming Languages*, pages 70–79, San Diego, California, January 13–15, 1988. ACM SIGACT-SIGPLAN, ACM Press.

[10] Gang Chen. Subtyping calculus of constructions. In *Proceedings of 22nd International Symposium, MFCS '97*, volume Lecture Notes in Computer Science 1295, pages 189–198. Springer-Verlag, 1997.

[11] Adriana Compagnoni and Healfdene Goguen. Decidability of higher-order subtyping via logical relations, December 1997.

[12] Adriana Compagnoni and Healfdene Goguen. Typed operational semantics for higher order subtyping. Technical Report ECS-LFCS-97-361, University of Edinburgh, July 1997. To appear in Information and Computation.

[13] Adriana Beatriz Compagnoni. *Higher-Order Subtyping with Intersection Types*. PhD thesis, Nijmegen Catholic University, 1995.

[14] Thierry Coquand. Pattern matching with dependent types. In *Proceedings on Types for Proofs and Programs*, pages 71–83, Båstad, Sweden, 1992.

[15] Thierry Coquand and Gérard Huet. The Calculus of Constructions. *Information and Computation*, 76(2/3):95–120, February/March 1988.

[16] Pierre-Louis Curien and Giorgio Ghelli. Coherence of subsumption, minimum typing and typechecking in $F_{\leq}$. *Mathematical Structures in Computer Science*, 2:55–91, 1992.

[17] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *JACM*, 40(1):143–184, 1993.

[18] Giuseppe Longo, Kathleen Milsted, and Sergei Soloviev. A logic of subtyping (extended abstract). In *Proceedings, Tenth Annual IEEE Symposium on Logic in Computer Science*, pages 292–299, San Diego, California, 26–29 June 1995. IEEE Computer Society Press.

[19] Zhaohui Luo. Coercive subtyping. *Journal of Logic and Computation*, 9(1):105–130, 1999.

[20] Ian A Mason. Hoare's Logic in the LF. Technical Report ECS-LFCS-87-32, LFCS, Department of Computer Science, University of Edinburgh, June 1987.

[21] Frank Pfenning. Refinement types for logical frameworks. In *Informal Proceedings of the 1993 Workshop on Types for Proofs and Programs*, pages 315–328, May 1993.

[22] Benjamin C. Pierce. *Programming with Intersection Types and Bounded Polymorphism*. PhD thesis, Carnegie Mellon University, December 1991. Available as School of Computer Science technical report CMU-CS-91-205.

[23] Benjamin C. Pierce. Bounded quantification is undecidable. In *Proceedings, Nineteenth ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 305–315, Albuquerque, New Mexico, January 19–22, 1992. ACM Press.

[24] Benjamin C. Pierce and David N. Turner. Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 4(2):207–247, April 1994.

[25] Donald Sannella, Stefan Sokołowski, and Andrzej Tarlecki. Toward formal development of programs from algebraic specifications: Parameterisation revisited. *Acta Informatica*, 29:689–736, 1992.

[26] Martin Steffen and Benjamin Pierce. Higher-order subtyping. In *IFIP Working Conference on Programming Concepts, Methods and Calculi (PROCOMET)*, June 1994.

[27] Jan Zwanenburg. *Object-Oriented Concepts and Proof Rules: Formalization in Type Theory and Implementation in Yarrow*. PhD thesis, Eindhoven University of Technology, 1999. PhD Thesis.