Information and Computation **168**(1):1-67, (2001)

Dependent types with subtyping and late-bound overloading

Giuseppe Castagna* and Gang Chen⋆

* *C.N.R.S.,* LIENS, *École Normale Supérieure, 45 rue d'Ulm, 75005 Paris, France*
⋆ *School of Computer & Information Sci., University of South Australia,The Levels, Australia*

E-mail: Giuseppe.Castagna@ens.fr; cisgc@cs.unisa.edu.au

Received June 23, 1998

We present a calculus with dependent types, subtyping and late-bound overloading. Besides its theoretical interest this work is motivated by several practical needs that range from the definition of logic encodings, to proof specialization and reuse, and to object-oriented extension of the SML module system.

The theoretical study of this calculus is not straightforward. While confluence is relatively easy to prove, subject reduction is much harder. We were not able to add overloading to any existing system with dependent types and subtyping, and prove subject reduction. This is why we also define here as by-product a new subtyping system for dependent types that improves previous systems and enjoys several properties (notably the transitivity elimination property). The calculus with overloading is then obtained as a conservative extension of this new system. Another difficult point is strong normalization, which is a necessary condition to the decidability of subtyping and typing relations. The calculus with overloading is not strongly normalizing. However, we show that a reasonably useful fragment of the calculus enjoys this property, and that its strong normalization implies the decidability of its subtyping and typing relations.

The article is divided into two parts: the first three sections provide a general overview of the systems and its motivations, and can be read separately; the remaining sections develop the formal study.

1. INTRODUCTION

In this article we show how to integrate in a unique logical system three different features: (first order) dependent types, subtyping, and late-bound overloading. We first describe each of these features and in the next section we illustrate the motivations of our work.

Dependent types.
Dependent types are types depending on terms. A classical example is given by arrays. Consider for example the arrays of characters. In programming languages there is not a type "`array of chars`" but rather a family of types `char[1]`, `char[2]`,..., where `char[`n`]` denotes the type of the character arrays of length n. Consider then the function

string_to_array that maps a string s into the array of its characters. Its domain type is string but its codomain type *depends* on the length of the string the function is applied to. More precisely, string_to_array is a function that maps a string s into an array of type char[*length*(s)]. By dependent types it is possible to express the type of this function as follows:

$$\texttt{string\_to\_array} \;:\; \pi s\colon \texttt{string}.\texttt{char}\,[\mathit{length}(s)]$$

In words, the typing judgment above expresses that string_to_array is a function that, when applied to a string s, returns a result of type char[*length*(s)] (π is a binder for the term variable s).

So dependent types allow to express a relationship between the input of a function and *the type* of its output. Dependent types are at the basis of many computer applications, notably automatic proof-checking —e.g., [HHP93, CAB^{+}86]— (since they offer the power of first order-logic), or rich module systems —e.g., [MQ85, Ler94]— (a module can export functions whose type is defined in the module itself, so the type of the result of a transformation of modules may depend on the module —on its type declaration part— the transformation is applied to).

Subtyping.

Subtyping is a binary relation over types. The introduction of a subtyping relation in a language greatly enhances its flexibility. Intuitively, a type S is a subtype of a type T (noted $S \leq T$) if all expressions of type S can be used in every context where an expression of type T is expected (for example, integer can be considered a subtype of real and char a subtype of string). The advantage of such a relation is that the code originally written for a given type can be reused for its subtypes (e.g. the function string_to_array can be applied to characters as well). This is obtained by adding to the typing rules the *subsumption* rule of [Car88]:

Subsumption
$$\frac{\Gamma \vdash M : S \qquad S \leq T}{\Gamma \vdash M : T}$$

that states that an expression that has type S is typed by every super-type of S, as well.

Late-bound overloading.

An overloaded function is a function that executes different code according to the type of its arguments. A typical example is the function + that, in several programming languages, performs arithmetic sum if applied to two numbers and concatenation if applied to two strings. Thus + can be thought of as the union of two different functions, arithmetic sum and string concatenation. More generally, every overloaded function consists of a set of functions, one for each possible combination of the types of its arguments. At type level this can be expressed by typing overloaded functions by sets of arrows. Thus for our example we have:

$$+ : \{\texttt{int} \times \texttt{int} \to \texttt{int}\;,\; \texttt{string} \times \texttt{string} \to \texttt{string}\}$$

In most programming languages selection of the code for an overloaded function call is performed at compile time: a pre-processor replaces every call of an overloaded function by the code that fits the type of the arguments. This discipline of selection is called *early binding*. In presence of subtyping the type of the arguments of a function may change (notably decrease) during computation. Therefore delaying the code selection to run-time may affect the semantics of programs. In particular we are interested in a *late binding* discipline that delays the selection as much as possible so that the selection is based on the best information about the type of the arguments. The interest of such a discipline is that, as shown in Section 2.2, it supports code reuse and incremental programming. This last point is also witnessed by object-oriented programming where (some special cases of) late-bound overloaded functions are better known as *multi-methods* or *generic functions* (see for example the languages Cecil [Cha92] and CLOS [DG87]).

In this article we show how to make these three features coexist in a unique formalism called $\lambda\Pi^{\&}$. This formalism can be considered as the natural extension with late-bound overloaded functions of $\lambda P_{\leq}$ [AC96b] (a calculus with dependent types and subtyping) or, equivalently, as the generalization to dependent types of $\lambda\&$ [CGL95] (a calculus with late-bound overloaded functions and subtyping). From a strictly technical point of view the main contribution of this work is the definition of a type discipline for late bound overloaded functions in the presence of dependent types. A subordinate contribution is the definition of a set of subtyping rules that defines the same typing relation as $\lambda P_{\leq}$ but enjoys much better properties which, among other things, make it prone to extensions.

Our work is not just an "exercice de style" where we try to put together some disparate functionalities for the sake of attempt. The logical difficulties and computational expressiveness of $\lambda\Pi^{\&}$ should be clear: computation depends on types, and possibly on *dynamic* types (because of late binding). Section 2 shows that, besides these logical aspects, $\lambda\Pi^{\&}$ answers also some practical needs. In Section 3 we give an overview of the whole system, by describing how we arrived at its definition. To that end we first introduce dependent types, we then add subtyping, and finally extend the result by late-bound overloaded functions. In Sections 4 and 5 we give the formal definition of the system and study its meta-theoretic properties: confluence, soundness, and strong normalization. In Section 6 we study some properties that are particular to our subtyping deduction system for dependent types, and in Section 7 we study the decidability of $\lambda\Pi^{\&}$. A conclusion ends our presentation.

In the section of overview we distinguish some text as "excursus". These excursi discuss some precise technical or practical points and can be skipped during the first reading.

2. MOTIVATIONS

There are three main motivations to our work. First and foremost, the need of both subtyping and overloading is quite felt in theorem proving, and their absence makes logic encodings much more difficult. Second, the use of late-bound overloading allows greater code reuse, introducing in some sense an object-oriented style in automatic proving. Last, dependent types constitute a theoretical basis of the SML module system; therefore our work may be useful to give a theoretical basis to object-oriented extensions of the SML module systems. Let us examine each motivation in more detail.

2.1. Logic encodings

The first order dependent type theory $\lambda\Pi$ [HHP93] (see Section 3.2 for a short formal presentation) has been taken as a Logical Framework for the specification of logical systems. For such a purpose, terms in this system are used to encode formulae in logic. Pfenning [Pfe93] demonstrates that in the absence of subtyping the representation of subsets of logical formulae is very cumbersome. This can be illustrated by the following example, which is adapted from the one in [Pfe93]. Consider the set of well-formed formulae of the propositional calculus characterized by the following abstract syntax:

$$F \ ::= \ A \mid \neg F \mid F \wedge F \mid F \vee F \mid F \Rightarrow F$$

Here A ranges over atomic formulae. This definition can be represented by the following set of typing declarations:

$$\begin{array}{lcl} F & : & \star \\ \neg & : & F \to F \\ \wedge & : & F \to F \to F \\ \vee & : & F \to F \to F \\ \Rightarrow & : & F \to F \to F \end{array}$$

Intuitively, $\star$ is the set of all types. Thus, $F : \star$ can be read as "F is a type". An example of formal encoding is

$$\Gamma, a : F, b : F \vdash \Rightarrow (\wedge ab)a : F$$

where Γ is a context containing the set of declarations defined above, and a, b are atomic formulae.

Now consider the subset of F defined as:

$$F_1 \ ::= \ A \mid \neg F_1 \mid F_1 \vee F_1$$

There are several ways to represent F_1. One way is to introduce a predicate on F, say $B : F \to \star$, such that $B(t)$ is true if and only if t is a formula of F_1.[1] Another way is to introduce a new type $F_1 : \star$. Both ways are awkward and lead to inefficient implementation of proof search (see [Pfe93]). To overcome the problem, Pfenning proposes to extend $\lambda\Pi$ with intersection types (see [BCDC83, CD80]) and *subsorting*. The latter, denoted by $<:$, can be viewed as a restricted form of subtyping. Then, it is possible to have a much better representation:

$$\begin{array}{rcll} A & <: & F_1 & \qquad\qquad A \text{ is a subsort of } F_1 \\ F_1 & <: & F & \qquad\qquad F_1 \text{ is a subsort of } F \\ \neg & : & (F \to F) \cap (F_1 \to F_1) & \\ \vee & : & (F \to F \to F) \cap (F_1 \to F_1 \to F_1) & \end{array}$$

Instead of using subsorting and intersection types, in this article, we propose to extend $\lambda\Pi$ with subtyping (denoted by $\leq$) and overloaded types (denoted by curly brackets). In the resulting system, that we dub $\lambda\Pi^{\&}$, the example above becomes:

$$\begin{array}{rcll} A & \leq & F_1 & \qquad\qquad A \text{ is a subtype of } F_1 \\ F_1 & \leq & F & \qquad\qquad F_1 \text{ is a subtype of } F \\ \neg & : & \{F \to F, F_1 \to F_1\} & \\ \vee & : & \{F \times F \to F, F_1 \times F_1 \to F_1\} & \end{array}$$

Subtyping and overloading are respectively richer than subsorting and intersections.

The difference between intersection and overloaded types is that a term belongs to the intersection $A \cap B$ if and only if it belongs both to A and B. While a term in the overloaded

[1] The complete representation for B includes the following declarations:

$$\begin{array}{lcl} A & : & F \to \star \\ \neg_1 & : & Bx \to B(\neg x) \end{array} \qquad\qquad \begin{array}{lcl} A_1 & : & Ax \to Bx \\ \vee_1 & : & Bx \to By \to B(\vee xy) \end{array}$$

where $A(t)$ is true if and only if t is an atomic formula.

type $\{A, B\}$ is the union of two distinct subterms, one belonging to A, the other to B. Note however that $\{A, B\}$ is defined only when both A and B are arrow types. In this case the overloading approach is somewhat more expressive since ultimately a term of an intersection type (of arrow types) can be considered as a special case of an overloaded term formed by a union of equal subterms.

Furthermore in Pfenning's system, decidability is obtained by defining subsorting over "sorts", which are refinements of types. Sorts cannot appear in labels of λ-abstractions so, as Pfenning points out, it is impossible to write functions with domains limited via subsorting. To overcome this weakness, Aspinall and Compagnoni have studied $\lambda P_{\leq}$ ([AC96b], see also Section 3.3.2), an extension of $\lambda\Pi$ with subtyping, which does not have such a drawback. However $\lambda P_{\leq}$ cannot express Pfenning's examples as it contains neither intersection types nor overloaded types. Here we define $\lambda\Pi^{\&}$. Since it has subtyping (but defined differently from [AC96b]) it does not have the weaknesses of the subsorting approach and thanks to overloading it can express Pfenning's examples.

Pfenning's study is developed within the proof environment Elf, an implementation of Edinburgh LF. Other groups studying dependent type theory based proof systems found the need of using subtyping, as well. All the motivating examples are similar to the Pfenning's one. An early work can be found in [Coq92] in the ALT group. LEGO, Coq, and Nuprl groups are studying implementations of abstract algebra, and all of them have proposed extensions of type theory by some sort of subtyping: ZhaoHui Luo [Luo96] has studied a "coercive subtyping" extension for LEGO; in the Nuprl group Jason Hickey [Hic95] has combined object-calculus and dependent types and proposed a form of subtyping based on the inheritance mechanism of objects; Courant in the Coq group is working on an extension of the Calculus of Construction by subtyping: $CC_{\leq}$ [Cou97]. More references to recent work are discussed in Section 6.3.

The interest in this area is mainly due to the scale. As said by ZhaoHui Luo[Luo96]: "the lack of useful subtyping mechanisms in dependent type theories [...] with inductive types and the associated proof development systems is one of the obstacles in their applications to large-scale formal development".

What we propose is to add not only subtyping but also late-bound overloading. In fact the association of these two features allows incremental and modular programming whose utility to large-scale problems has been widely demonstrated by object-oriented languages.

2.2. Program (proof) specialization

Consider again the types F and F_1 defined in the previous section. Since $F_1 \leq F$, then $F \to \mathit{Bool} \leq F_1 \to \mathit{Bool}$. [2] Hence, a decision function p for propositional logic, which is of type $F \to \mathit{Bool}$, is also a decision function for F_1 formulae (i.e., it is of type $F_1 \to \mathit{Bool}$).

So subtyping is a first ingredient for code reusing since it allows to use p on arguments of type F_1 even if p has been written for arguments of type F. However, subtyping provides a limited form of reusing: it just makes some code more polymorphic. A breakthrough for code reusing (brought forward by object-oriented languages) is code specialization. Consider again the decision function p. It is well known that such a function p is NP-hard. However, by the specific structure of F_1, it is possible to construct a polynomial decision function p_1 for F_1. A clever way to define a general decision function `dec` is then use

[2] The subtyping rule for arrow types [Car88] states that if $A_2 \leq A_1$ and $B_1 \leq B_2$ then $A_1 \to B_1 \leq A_2 \to B_2$ (see [Cas95] for a detailed discussion).

p_1 for F_1 formulae and p otherwise. A natural way to obtain it is to define our decision function as an overloaded function formed by the two terms p and p_1. In the notation we use in this work an overloaded function composed by the terms p and p_1 is written as $p\&p_1$. So `dec`$\equiv p\&p_1$. The type of `dec` is the union of the types of the composing terms, that is $\{F \to \mathit{Bool}, F_1 \to \mathit{Bool}\}$. The (overloaded) function $p\&p_1$ automatically chooses the appropriate sub-term to execute (i.e., either p or p_1) according to the type of its argument (that is, according to the form of the formula to decide) . The use of late binding ensures that the most efficient function will be always selected even in the case that the most specific form of the formula is not, or cannot be, statically determined. The user is released from writing branch selection code. What he has to do is just to declare the subtyping structure.

PRACTICAL EXCURSUS. *Note that this could be done in an incremental way. We could have first defined just F with the decision function* `dec := ` p *and decided only later to consider the F_1-formulae.*

By declaring $F_1 \leq F$ we can use for F_1-formulae all code written for F-formulae. By specializing `dec := dec`$\&p_1$, *every code that uses* `dec` *is specialized as well. Thus all code for F is automatically specialized (and, thus, reused) for F_1. However, this situation is more complex than the one we present in this article, since it requires* `dec` *to be* dynamically extensible *(whence the use of "*`:=`*"). This is discussed in Section 2.1 of [CGL93]. In this article we focus on the logical aspects of the system and we do not deal with this issue that looks more related to implementation.* □

2.3. Extension of the SML module system

In the SML module system [MTH90] a module may export both some types and the operations defined for these types. Thus the *type* of the operation components of a module—and, thus, the type of the module itself—may depend on the *value* of the type components of the module. Since we are in presence of types that depend on values, then a classical approach to characterize the SML module system is to use first order dependent types [MQ86, MH88, Ler94, HL94].

Modules are handled by *functors*. Functors are functions that transform modules into other modules and that are subtype polymorphic (intuitively, a functor defined for modules that export some given components works also on modules that export more components). Functors can be considered as modules parametrized by some other modules. One of the criticisms to the SML module system is that although it has subtyping, it is not possible to perform code reuse and specialization as done in "object-oriented programming". In order to make it possible, Aponte and Castagna defined in [AC96a] an extension of the SML module system with late-bound overloaded functors. The starting system is the one of Leroy [Ler94], and the addition of late-bound overloaded functors allows to choose the most specific transformation of a module according to its type. The result is a module programming language whose style is very similar to the one of CLOS where the "generic functions" ("generic functors" in this case) operate on modules rather than on objects, and their behavior can be incrementally specialized as long as new modules types (signatures, in the SML terminology) are defined. The idea can be illustrated by the following example.

Consider a dictionary module `mkDict` parameterized by a tree module t of type `Tree`:

```
functor mkDict(t:Tree):Dict = struct … end
```

The type of such a functor is a first order dependent type:

$$\texttt{mkDict} : \pi t\text{: Tree.Dict}(t)$$

the dependency is necessary since the type of the result depends on (the type of the elements of) the argument t of the functor (we stress this dependency by writing Dict(t)). If a signature `OrdTree` for ordered trees is available, and `OrdTree`≤`Tree` then, by subtyping, it is possible to feed `mkDict` by ordered trees to make new dictionaries. It is also possible to define a new functor `mkOrdDict`: πt: OrdTree.Dict(t), that provides an optimal search operation by keeping the ordered tree balanced. However all the code that still uses `mkDict` will continue to produce inefficient code for ordered trees.

The solution is to overload the functor `mkDict` by the more efficient code for ordered trees (in [AC96a] this is performed by the command `extend functor MkDict by MkOrdDict`), so that the functor will execute two different pieces of code according to whether the argument module implements a `Tree` or an `OrdTree`. In other words, the `mkDict` will be an overloaded functor of type $\{\pi t$:Tree.Dict(t) , πt:OrdTree.Dict(t) $\}$ An outline of the code of this example can be found in Appendix A.2. For more details the reader can refer to [AC96a].

One of the problems with this system is to prove its type soundness. The standard technique for type soundness is to prove the subject reduction property (reductions preserve types). Unfortunately the subject reduction property does not hold for Leroy's system and, therefore, it does not hold for the [AC96a] system either. While Leroy was able to prove the soundness of his system by semantic tools (he uses a translation into a system with dependent types, second-order existential types, and Σ-types [Ler94]), his proof does not extend to overloaded functors whose theoretical bases are not established. $\lambda\Pi^{\&}$ is a first step towards establishing these bases and proving soundness of the work in [AC96a].

3. INFORMAL DESCRIPTION

In this section we give an intuitive description of our system $\lambda\Pi^{\&}$. At the risk of some redundancy, we prefer to defer the formal definition of $\lambda\Pi^{\&}$ to Section 4 and show here, step by step, the path that leads to the definition of the whole system. So we start by defining dependent types, that is the system $\lambda\Pi$ (§ 3.2). Then, we introduce subtyping for dependent types, that is the system $\lambda\Pi_{\leq}$. Even if $\lambda\Pi_{\leq}$ owes a lot to the Aspinall and Compagnoni system $\lambda P_{\leq}$, we show that $\lambda P_{\leq}$ does not fit our purposes since, because of its formalization, it is not prone to extension and $\lambda\Pi_{\leq}$ is needed (§ 3.3). Finally, we introduce overloading for the previous system, yielding the system $\lambda\Pi^{\&}$ (§ 3.4). We conclude this section by several examples, and by summarizing all the technical results that will be shown in the rest of the paper.

3.1. A brief introduction to dependent type theory

Types are used to classify terms, but with dependent types we have seen that types and terms are not completely distinct. For example in Section 1 we described the function `string_to_array` where the type of the result depended on the input of the function. This was expressed by a type of the form $\pi s\colon \texttt{string}.A(s)$. We have also seen *type families*, such as the family of arrays of characters $\{\texttt{char[1]},\ \texttt{char[2]}, \ldots\}$. Type families can be considered as mappings from terms to types; for example the above family of arrays of characters corresponds to the map $n \mapsto \texttt{char}[n]$. Since we use $\star$ to classify types then this mapping can be "typed" by the "kind" $\Pi n\colon \texttt{nat}.\star$.

Type families can be expressed by Λ-notation. So for example

$$\Lambda n{:}\,\texttt{nat}.\texttt{char[}n\texttt{]} \;:\; \Pi n{:}\,\texttt{nat}.\star$$

denotes the type family described right above. By β-reduction then $(\Lambda n{:}\,\texttt{nat}.\texttt{char[}n\texttt{]})(3)$ is the type of the arrays of characters of size 3, that is `char[3]`.

More generally, we are considering a dependent type system where terms are classified by types and types (or, more generally, type families) are classified by kinds. Types are either atomic types (e.g., `int`), applications of type families (if they have the kind $\star$), or π-types of form $\pi x{:}A.B$ (that are used to type λ-abstractions). In particular, π-types are the generalization of arrow types of simply-typed λ-calculus: $A \to B$ is the special case of $\pi x{:}\,A.B$ where x does not appear free in B.

From the point of view of the formula-as-type analogy, the introduction of dependent types brings significant progress with respect to simply typed lambda calculus. In the latter case, only propositional formulae can be represented by types, while dependent types make first order quantification representable as well. As a result, many logical systems can be encoded in systems based on dependent types, as done in LF, the Edinburgh Logical Framework [HHP93].

3.2. Dependent types: the system $\lambda\Pi$

The system $\lambda\Pi$ [HHP93] is the pure first order dependent type system (a different version of the system is called λP [Bar92]) . It is the core of Edinburgh Logical Framework. Our presentation of $\lambda\Pi$ is mainly based on [HHP93]. There are four syntactic categories:

Terms	M	$::=$	$x \mid \lambda x{:}A.M \mid MM$
Types	A	$::=$	$\alpha \mid \pi x{:}A.A \mid \Lambda x{:}A.A \mid AM$
Kinds	K	$::=$	$\star \mid \Pi x{:}A.K$
Contexts	Γ	$::=$	$<> \mid \Gamma, x : A \mid \Gamma, \alpha : K$

1. A term (denoted by $M, N, \dots$) is either a term variable (denoted by $x, y, z, ...$), an abstraction or an application.

2. A type (denoted by $A, B, C, \dots$) is either an atomic type (denoted by α), a π-type of the form $\pi x{:}A.B$ or a type application AM or a type family $\Lambda x{:}A.M$.

3. A kind is either the constant $\star$ representing the collection of all types, or $\Pi x{:}A.K$ which classifies type families (of the form $\Lambda x{:}A.B$ where B lives in the kind K). Thus,the general form of a kind is $\Pi x_1{:}A_1..x_n{:}A_n.\star$ with $n \geq 0$.

4. A context is an ordered list of typing assignments of the form $x{:}\,A$, and of kinding assignments of the form $\alpha{:}\,K$. If $x{:}\,A$ appears in Γ then we say that $x \in \mathit{Dom}(\Gamma)$ and we use $\Gamma(x)$ to denote A. If $\alpha{:}K$ appears in Γ then we say that $\alpha \in \mathit{Dom}(\Gamma)$ and we use $\mathit{Kind}_\Gamma(\alpha)$ to denote K.

The atomic types α play the role of (dependent) type constants[3]; typical examples of type constants are `int`, `nat`, `bool` (all declared of kind $\star$), and `char[ ]` (of kind $\Pi n{:}\,\texttt{nat}.\star$). We use $M[x := N]$ (resp., $B[x := N]$) to denote the substitution of N for every free occurrence of x in term M (resp., in type B). β-reduction, denoted by $\to_\beta$, is the *compatible closure*

[3] In [Bar92] and [AC96b] the α's are called type *variables*. This may be misleading since although α's are declared in contexts, they cannot be abstracted.

Context Formation

F-EMPTY $$\frac{}{<> \vdash \star}$$

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

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

F-Π $$\frac{\Gamma, x : A \vdash K}{\Gamma \vdash \Pi x{:}A.K}$$

Typing

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

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

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

T-CONV $$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A =_\beta B}{\Gamma \vdash M : B}$$

Kinding

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

K-π $$\frac{\Gamma, x : A \vdash B : \star}{\Gamma \vdash \pi x{:}A.B : \star}$$

K-Λ $$\frac{\Gamma, x : A \vdash B : K}{\Gamma \vdash \Lambda x{:}A.B : \Pi x{:}A.K}$$

K-APP $$\frac{\Gamma \vdash A : \Pi x{:}B.K \quad \Gamma \vdash M : B}{\Gamma \vdash AM : K[x := M]}$$

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

FIG. 1. The $\lambda\Pi$ type system

(see [Bar84]) of the union of the following two notions of reduction:

$$\begin{array}{l}(\lambda x{:}A.M)N \rightarrow_{\beta_1} M[x := N] \\ (\Lambda x{:}A.B)N \rightarrow_{\beta_2} B[x := N]\end{array}$$

β-conversion, denoted by $=_\beta$ is the equivalence relation generated from β-reduction, that is, the reflexive, symmetric, and transitive closure of $\rightarrow_\beta$.[4]

The abstract syntax above defines pre-terms, pre-types, pre-kinds, and pre-contexts, namely possibly not well-formed terms, types, kinds, and contexts. Well-formed terms, types, kinds, and contexts are determined by the following four judgments:

[4] Since β-reduction and β-conversion are *compatible* relations, then they are defined on terms, types, *and* kinds (since in the last one both types and terms may occur).

$\Gamma \vdash \star$	Γ is a well-formed context
$\Gamma \vdash K$	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

We write $\Gamma \vdash J$ for an arbitrary judgment of the form $\Gamma \vdash K, \Gamma \vdash A : K$ or $\Gamma \vdash M : A$. The rules for deriving the judgments in $\lambda\Pi$ are in Figure 1.

3.3. Adding subtyping to $\lambda\Pi$: the system $\lambda\Pi_{\leq}$

The addition of subtyping to an existing type system is usually performed in a standard two-step process. First, a subtyping relation $\leq$ is defined on the (well-formed) types of the system. Then the subsumption rule is added to the typing rules (and when the conversion rule is present it replaces it). We already said that the subsumption rule states that if a term is typed by some type A, then it is also typed by every super-type of A. Usually this rule has the form we saw in the Introduction:

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

Note that, as the subtyping relation is defined on (well-formed) types, then no kinding judgment is required in this rule: if A and B are in subtyping relation, then they are well-formed.

In this work, for reasons that we explain at length in the Section 3.3.2, we need to define subtyping in a different manner.

More precisely, we do not define the subtyping relation on the $\lambda\Pi$'s types and do not substitute the subsumption rule above for the T-CONV rule of the previous section. Instead, we define the subtyping relation on the $\lambda\Pi$'s *pre-types* (that may be not well-formed) and replace T-CONV by the following subsumption rule

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

where $\Gamma \vdash A, B \colon K$ is a shorthand for "$\Gamma \vdash A \colon K$ and $\Gamma \vdash B \colon K$". This rule states exactly the same property as the generic subsumption rule above. However, it has two extra kinding premises that are made necessary by the fact that, here, $\leq$ is defined on all pre-types.

3.3.1. The subtyping relation

The subtyping relation on the pre-types arises from a subtyping relation on atomic types. This relation for atomic types is declared in a context Γ and lifted up to all types by the rules of Figure 2. More precisely, subtyping declarations occur in a context Γ under the form of bounded kind assignment $\alpha \leq A : K$. In that case we say that $\alpha \in \mathit{Dom}(\Gamma)$ and that α is *bounded* in Γ; we also use $\Gamma(\alpha)$ to denote A, and still use $\mathit{Kind}_\Gamma(\alpha)$ to denote K.

In summary, the system $\lambda\Pi_{\leq}$ is defined by the rules in Figure 2 plus all the rules for $\lambda\Pi$ defined in the previous section, but where Γ may contain bounded kind assignments and T-SUB is substituted for T-CONV. [5]

Since the rules in Figure 2 do not contain any kinding judgment, then the induced subtyping relation is defined on all pre-types. The restriction of this relation to types (that is, to well-kinded pre-types) has the usual general meaning: a term of a given type can be

[5] We must also add an obvious formation rule for type contexts containing subtyping constraints (see Appendix A.1).

S-ApT	$\dfrac{\Gamma \vdash \Gamma(\alpha)M_1..M_n \leq A}{\Gamma \vdash \alpha M_1..M_n \leq A}$
S-ApR	$\dfrac{M_1 =_\beta M_1' \ \cdots \ M_n =_\beta M_n'}{\Gamma \vdash \alpha M_1..M_n \leq \alpha M_1'..M_n'}$
S-π	$\dfrac{\Gamma \vdash A' \leq A \qquad \Gamma, x : A' \vdash B \leq B'}{\Gamma \vdash \pi x{:}A.B \leq \pi x{:}A'.B'}$
S-Λ	$\dfrac{A' =_\beta A \qquad \Gamma, x : A' \vdash B \leq B'}{\Gamma \vdash \Lambda x{:}A.B \leq \Lambda x{:}A'.B'}$
S-ApSL	$\dfrac{\Gamma \vdash B[x := M_1]M_2..M_n \leq C}{\Gamma \vdash (\Lambda x{:}A.B)M_1..M_n \leq C}$
S-ApSR	$\dfrac{\Gamma \vdash C \leq B[x := M_1]M_2..M_n}{\Gamma \vdash C \leq (\Lambda x{:}A.B)M_1..M_n}$

FIG. 2. $\lambda\Pi_\leq$ subtyping rules

safely used wherever a term of a super-type is expected. Let us comment each rule starting from the simplest ones:

- The S-π rule is the generalization of subtyping rule for arrow types. It is contravariant on the domains and covariant on codomains. However, since the bound variable x can appear free in the codomains, then the codomains are compared under the assumption that x belongs to the domain common to both types, that is, the smaller one.
- The S-Λ rule "subtypes" type families. Recall that type families are functions from terms to types. This rule states that two such functions are compared pointwise. (As a matter of fact this rule is useless in $\lambda\Pi_\leq$ and should be omitted: see the Excursus in Section 4.4).
- The S-ApSL and S-ApSR rules state that subtyping is invariant by β_2 head reductions: to deduce that a β_2 head redex is in a subtyping relation we must deduce it for its reductum.
- The S-ApR states that $\leq$ is reflexive on atomic types. The reflexivity is extended pointwise to all possible applications of the atomic type.
- The S-ApT (combined with reflexivity) performs the transitive closure of the subtyping declarations. Intuitively in order to prove that $\Gamma \vdash \alpha \leq \Gamma(\Gamma(\Gamma(\alpha)))$ three S-ApT rules topped by a S-ApR rule must be used. As for S-ApR the relation is extended pointwise to possible applications.

Note that not all the assignments in a context Γ equally contribute to the definition of subtyping. Only bounded kind assignments $\alpha \leq A{:}K$ really matter, since they are used by the rule S-ApT. Kinding assignments $\alpha{:}K$ are handled by the rules S-π and S-Λ only to ensure that for every subtyping rule the well-kindedness of the types appearing in the conclusion under a given context implies the well-kindedness of the types appearing in the premises under the corresponding contexts.

S^{AC}*-var* $$\frac{\Gamma \vdash \star \qquad \alpha \textit{ bounded in } \Gamma}{\Gamma \vdash^{AC} \alpha \preceq \Gamma(\alpha)}$$

S^{AC}-π $$\frac{\begin{array}{c}\Gamma \vdash \pi x{:}\, A.B, \pi x{:}\, A'.B' : \star \\ \Gamma \vdash^{AC} A' \preceq A, \qquad \Gamma, x : A' \vdash^{AC} B \preceq B'\end{array}}{\Gamma \vdash^{AC} \pi x{:}A.B \preceq \pi x{:}A'.B'}$$

S^{AC}-Λ $$\frac{\begin{array}{c}\Gamma \vdash^{AC} \Lambda x{:}\, A.B, \Lambda x{:}\, A'.B' : K \\ A' =_\beta A \qquad \Gamma, x : A' \vdash^{AC} B \preceq B'\end{array}}{\Gamma \vdash^{AC} \Lambda x{:}A.B \preceq \Lambda x{:}A'.B'}$$

S^{AC}*-app* $$\frac{\Gamma \vdash^{AC} A \preceq B \qquad \Gamma \vdash AM, BM : K}{\Gamma \vdash^{AC} AM \preceq BM}$$

S^{AC}*-conv* $$\frac{\Gamma \vdash A, B : K \qquad A =_\beta B}{\Gamma \vdash^{AC} A \preceq B}$$

S^{AC}*-trans* $$\frac{\begin{array}{c}\Gamma \vdash A, B, C : K \\ \Gamma \vdash^{AC} A \preceq B \qquad \Gamma \vdash^{AC} B \preceq C\end{array}}{\Gamma \vdash^{AC} A \preceq C}$$

FIG. 3. $\lambda P_{\leq}$ subtyping rules

3.3.2. A different presentation of subtyping (comparison with [AC96b])

Apart from S-π and S-Λ, the remaining rules of $\lambda\Pi_{\leq}$ are quite technical and do not let the reader to grasp the intuition of the subtyping relation. So we decided to add this section in order to provide some intuition. However, this section is not necessary to the development of this work: it is not used for defining $\lambda\Pi^{\&}$ and can be skipped at first reading (as signaled by the detour panel).

In order to provide the reader with the intuition underlying subtyping, we describe a set of subtyping rules different from the ones of $\lambda\Pi_{\leq}$. These rules define a subtyping relation "equivalent" (in the sense we precise later on) to the one of $\lambda\Pi_{\leq}$. The rules are shown in Figure 3. Apart from some minor differences[6], these rules are those used by David Aspinall and Adriana Compagnoni [AC96b] to define the system $\lambda P_{\leq}$, that is one of the best subtyping system for $\lambda\Pi$ available in the literature. In order to differentiate this second system from all the systems that are the contribution of this article, we use lowercase italicized names for rules and put *AC* scripts all over. We also use a different symbol, $\preceq$, to denote the new relation. Let us comment the $\lambda P_{\leq}$ subtyping rules:

- The S^{AC}*-var* rule deduces the subtyping declarations contained in Γ.
- The S^{AC}-π and S^{AC}-Λ have the same meaning as the corresponding rules in $\lambda\Pi_{\leq}$.
- The rule S^{AC}*-app* is a direct consequence of the interpretation of $S^{AC}-\Lambda$: if two functions are pointwise related then the images of a same point are related as well.

[6] There are some extra kinding judgments and the S-Λ rule here is more general than the one in [AC96b]. See Footnote 10 for an example.

- Finally the rules S^{AC}-*conv* and S^{AC}-*trans* state that $\preceq$ is a pre-order, that is a reflexive and transitive relation.

The two sets of rules in Figures 2 and 3 define the same subtyping relation. This is stated by the following property proven in Section 6.3:

PROPERTY 3.1. *For every A, B such that $\Gamma \vdash A, B : K$ we have*

$$\Gamma \vdash^{AC} A \preceq B \quad \Leftrightarrow \quad \Gamma \vdash A \leq B$$

(where by $\Gamma \vdash J$ we mean that the judgment is provable)

It is very important to notice that the property above says that the two sets of rules define the same relation on $\lambda\Pi$'s types ($\preceq$ is not defined on pretypes), but it does *not* say that the two sets of rules are completely equivalent, namely that it is possible to use either of them without any difference. In particular, while the rules of $\lambda\Pi_{\leq}$ constitute the core rules of this article, those of $\lambda P_{\leq}$ are inadequate to the purposes of this work. Indeed, the rules in Figure 2 satisfy two crucial properties that those in Figure 3 do not:

1. They do not use kinding judgments. This makes them prone to extension. Indeed, recall that we want to extend this system with overloaded types. As we shall see later, the kinding of overloaded types depends on the subtyping relation. So it is important to have the definition of subtyping separated from the one of kinding since, otherwise, we would have a circularity that is very difficult to handle.
2. They do not use the transitivity rule (which is an *admissible* rule, i.e., it is a consequence of the other rules[7]). This, intuitively, implies that the addition of new types and new rules to this type system is likely to yield a *conservative extension*[8] (the explicit use of the transitivity rule may cause a problem with conservativity since this rule does not satisfy the subformula property). So we have extensions that do not interfere with the original theory, independently from its definition.

For these reasons, the definitions of this article never use the rules in Figure 3 and they can (actually, must) be ignored. However, the reader can use them as a cue to understand the subtyping relation and draw intuition about it. But he must also be aware that in case of extension the equivalence of two set of rules may be lost.

TECHNICAL EXCURSUS. *The reader may be puzzled by the fact that of two sets of rules defining the same relation, one set is completely inadequate to certain purposes that the other fits. Apart from the fact that $\lambda\Pi_{\leq}$ and $\lambda P_{\leq}$ do not define the same subtyping relation ($\preceq$ is not defined for pretypes), this "anomaly" mainly concerns the possible extensions of the rules. The fact that two sets of rules define the same relation does not imply that this holds for every possible extension of these sets. As a trivial example consider the system formed just by the symmetry rule (that states that if (a, b) belongs to the relation then (b, a) belongs to it, too) and the system with no rules at all. The two deduction systems define the same relation (the empty relation), but it is clear that every non-symmetric extension of these sets of rules will not define the same relation.*

[7] Given a set $\mathscr{S}$ of deduction rules a (new) rule is *admissible* if for every instance of the rule it is possible to prove by the rules of $\mathscr{S}$ that its premises imply its consequence. Furthermore, the rule is *derivable*—or *derived*—if the rule can be obtained by composing some rules of $\mathscr{S}$.

[8] Given a language $\mathscr{L}$ and a notion of derivability $\vdash$ on $\mathscr{L}$, a *theory* $\mathscr{T}$ is a collection of sentences in $\mathscr{L}$ with the property that if $\mathscr{T} \vdash \varphi$ then $\varphi \in \mathscr{T}$. A theory $\mathscr{T}'$ is an *extension* of a theory $\mathscr{T}$ if $\mathscr{T} \subseteq \mathscr{T}'$. $\mathscr{T}'$ is a *conservative extension* of $\mathscr{T}$ if $\mathscr{T}' \cap \mathscr{L} = \mathscr{T}$.

If we do not consider extensions and we stick to the actual language, then the relation between $\lambda\Pi_{\leq}$ and $\lambda P_{\leq}$ is quite typical. $\lambda\Pi_{\leq}$ is the algorithmic version of $\lambda P_{\leq}$ and Property 3.1 is the classic formulation of the soundness and completeness of the algorithmic subtyping[9]. Note that because of Property 3.1, $\lambda\Pi_{\leq}$ constitutes an important improvement over the subtyping algorithm of [AC96b] for which this property does not hold. In fact, for the algorithm defined in [AC96b] only the following implications hold

$$\Gamma \vdash^{AC} A \preceq B \quad \Rightarrow \quad \Gamma \vdash^{AC}_{\mathscr{A}} (A)^{\beta_2} \preceq (B)^{\beta_2}$$
$$\Gamma \vdash^{AC}_{\mathscr{A}} (A)^{\beta_2} \preceq (B)^{\beta_2} \quad \Rightarrow \quad \Gamma \vdash^{AC} (A)^{\beta_2} \preceq (B)^{\beta_2}$$

(where $\vdash^{AC}_{\mathscr{A}}$ denotes deduction in the Aspinall-Compagnoni's algorithm and $(A)^{\beta_2}$ denotes the β_2-normal-form of A) which are weaker than Property 3.1.[10]

However, the interest of $\lambda\Pi_{\leq}$ is not confined to this aspect. While Property 3.1 is an interesting property in the context of $\lambda P_{\leq}$, it becomes crucial in the context of this article, since it frees the system from the transitivity rule without affecting its expressiveness. And while we know how to add late-bound overloaded functions to $\lambda\Pi_{\leq}$, the corresponding extension of $\lambda P_{\leq}$ is still an open problem. □

METHODOLOGICAL EXCURSUS. *Our first attempt for this work was to add rules for overloaded types to the Aspinall-Compagnoni system. Thus we obtained a circularity among the definitions of context formation, kinding, typing and subtyping. This complicated the proofs. For example the proof of the "classical" substitution lemma for subject-reduction is done in [AC96b] by simultaneous induction over the four forms of judgment. This same technique did not work in our case because the induction hypothesis does not suffice to prove that the conditions for well-kinding of overloaded types (see Section 3.4.4) are preserved by substitution.*

In $\lambda\&$, this proof does not have that problem since the subtyping system does not depend on the kinding system. The subtyping relation is defined over pre-types, that is, expressions that may not be well-kinded. Thus our second try was to erase the kinding premises from the Aspinall-Compagnoni subtyping system. This did not work either because of the transitivity rule that became:

$$\frac{\Gamma \vdash A \leq B \qquad \Gamma \vdash B \leq C}{\Gamma \vdash A \leq C}$$

but in presence of such a rule it is not possible to deduce the well-kinding of B from the well-kinding of A and C (and so we cannot ensure that the derivations of judgements with well-kinded types contain only well-kinded types). In $\lambda\&$ the transitivity rule can be eliminated since the structural subtyping rules extend the transitivity of subtyping on atomic types to higher types. But if we remove the transitivity rule from the Aspinall-Compagnoni system we do not obtain an equivalent system. And we cannot use the algorithmic system of

[9] As customary, we define the subtyping algorithm by a set of subtyping rules that satisfy the subformula property. When the set of rules is "syntax directed" (i.e., there is a one-to-one correspondence between provable judgments and proof trees), then the algorithm is deterministic (e.g. see §1.3 of [Cas97] for details). The set of subtyping rules of $\lambda\Pi_{\leq}$ can be straightforwardly turned into a deterministic algorithm by adding to the [S-ApSR] rule the condition $C \not\equiv (\Lambda x{:}A'.B')M'_1..M'_m \wedge C \not\equiv \alpha M'_1 \dots M'_m$ and to the rule [S-ApT] the condition $A \not\equiv \alpha M'_1 \dots M'_n$ (note the indexes). The system with these conditions is equivalent to the one without the conditions as proved in Section 6.4.

[10] For example if $A_i \to_{\beta_i} B_i$ $(i = 1, 2)$, then $\Gamma \vdash^{AC} A_2 \preceq B_2$ (and thus $\Gamma \vdash A_2 \leq B_2$) but $\Gamma \not\vdash^{AC}_{\mathscr{A}} A_2 \preceq B_2$. Furthermore, in the original definition of subtyping in [AC96b] one also has $\Gamma \vdash^{AC}_{\mathscr{A}} \pi x{:}\, A_1.C \preceq \pi x{:}\, B_1.C$ (and $\Gamma \vdash \pi x{:}\, A_1.C \leq \pi x{:}\, B_1.C$) but $\Gamma \not\vdash^{AC} \pi x{:}\, A_1.C \preceq \pi x{:}\, B_1.C$.

Aspinall and Compagnoni since it is defined on β_2 normalized types, not on types (whence the failure of Property 3.1 for this system).

Thus we decided to define a new transitivity free set of rules that did not use kinding judgments and that defined on (well-kinded) types the same system as Aspinall-Compagnoni one. The result of this attempt is the definition of $\lambda\Pi_{\leq}$. $\square$

3.4. Adding overloading to $\lambda\Pi_{\leq}$: the system $\lambda\Pi^{\&}$

In this section we give the description of the complete system, called $\lambda\Pi^{\&}$, which includes dependent types, subtyping, and late-bound overloaded functions. $\lambda\Pi^{\&}$ is obtained by adding late-bound overloaded functions to $\lambda\Pi_{\leq}$. Equivalently, it can be also considered as the generalization to dependent types of the $\lambda\&$-calculus of overloaded functions described in this same journal [CGL95], and revised in [Cas97].

3.4.1. Overloaded functions

An overloaded function is a function that executes a different code according to the type of its argument. Thus an overloaded function is formed by a set of ordinary functions (i.e. λ-abstractions), each one defining a different code (we call it *branch*) of the function. We follow the ideas of the $\lambda\&$-calculus and glue these functions together into an overloaded one by the symbol & (whence the name of the calculus). Thus, we add to the $\lambda\Pi_{\leq}$'s terms the term

$$(M\&N)$$

which intuitively denotes an overloaded function with two branches, M and N, one of which will be selected according to the type of the argument. We must distinguish ordinary function application from the application of an overloaded function since they constitute different mechanisms[11]. Thus we use "•" to denote "overloaded application" and "·" or simple juxtaposition for the usual application.

We build overloaded functions as lists, starting with an *empty* overloaded function, denoted by ε, and concatenating new branches by means of &. Thus, an overloaded function is a list of ordinary functions and, in the term above, M is an overloaded function while N is an ordinary function, (a branch of the resulting overloaded function). Therefore, an overloaded function with n branches $M_1, M_2, \dots M_n$ can be written as

$$((\dots((\varepsilon\&M_1)\&M_2)\dots)\&M_n).$$

The type of an overloaded function is the set of the types of its branches. Thus, if $M_i : \pi x{:}\,A_i.B_i$ then the overloaded function above has type

$$\{\pi x{:}\,A_1.B_1\ ,\ \pi x{:}\,A_2.B_2, \dots, \pi x{:}\,A_n.B_n\}$$

and if we apply this function to an argument N of type A_j, then the selected branch is M_j. That is

$$(\varepsilon\&M_1\&\dots\&M_n)\bullet N \quad\twoheadrightarrow\quad M_j{\cdot}N \tag{1}$$

where $\twoheadrightarrow$ means "reduces in zero or more steps" (the introduction of subtyping will require some restrictions to this reduction).

[11] The former is implemented by substitution, while the latter is implemented by selection.

3.4.2. Subtyping

If we were extending $\lambda\Pi$ by overloaded functions we could (nearly) stop here. But we are extending $\lambda\Pi_{\leq}$, so we have subtyping as well. Thus, we have to define the subtyping relation for the new overloaded types. The definition follows from the observation that an overloaded function can be used in place of an overloaded function of different type when, for each branch that can be selected in the latter, there is at least one branch in the former that can replace it. Thus, an overloaded type S, i.e., a set of π-types, is smaller than another overloaded type T if and only if for every type in T there is at least one type in S smaller than it. Formally, we add to the rules of $\lambda\Pi_{\leq}$ (without the S-Λ rule) the following subtyping rule:

S-OVER $$\frac{\forall j \in J\ \exists i \in I \qquad \Gamma \vdash \pi x{:}A_i.B_i \leq \pi y{:}C_j.D_j}{\Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\in I} \leq \{\pi y{:}C_j.D_j\}_{j\in J}}$$

Equivalently, in order to prove that $\{\pi x{:}A_i.B_i\}_{i\in I}$ is a subtype of $\{\pi y{:}C_j.D_j\}_{j\in J}$ one has to show that there exists a total map ϕ from J to I such that for every $j \in J$ it is provable that $\pi x{:}A_{\phi(j)}.B_{\phi(j)} \leq \pi y{:}C_j.D_j$.

Another consequence of using subtyping is that in a reduction like (1), the type of N may match none of the A_i , but rather be a subtype of some of them. In this case, we choose the branch whose A_i "best approximates" the type, say A, of N. That is, we select the branch j such that $A_j = \min_{i=1..n}\{A_i \mid A \leq A_i\}$. A restriction on the formation of overloaded types and the type system will ensure the existence of this minimum (Section 3.4.4).

It is well-known that in presence of subtyping a computation may change—precisely, may decrease—the type of a term.[12] If the term at issue is the argument of an overloaded function, then different degrees of computation may lead to different branch selections. Thus we have to determine when the selection for an overloaded application must be performed. We follow a *late selection* (or *late binding*) discipline since it allows a high level of code reuse and an incremental style of programming (see §2.1 of [Cas97]). Therefore we impose that a reduction such as (1) can be performed if and only if N is closed (i.e., without free term variables) and in normal form (i.e., it cannot be reduced any more).

3.4.3. Annotations

Determining a selection discipline is not enough to make the extension a coherent calculus. We also have to *freeze* the type of overloaded functions. Consider the following example: let M be a term of type $\pi x{:}\, C.B$, a subtype of $\pi x{:}\, A.B$. Then the body of the following function (we omit the leading ε)

$$\lambda y{:}\,(\pi x{:}\, A.B).(y\ \& M)$$

has type $\{\pi x{:}\, A.B, \pi x{:}\, C.B\}$. If we apply this function to M itself, this application reduces to $M\&M$ of type $\{\pi x{:}\, C.B, \pi x{:}\, C.B\}$, which is nonsensical, since both branches are defined for arguments of the same type (so there no longer is a "best approximating" branch). Therefore, in order to record that the first branch was intended for arguments of type A and the second one for arguments of type C, we annotate the & by the (intended) type of the term:

$$\lambda y{:}\,(\pi x{:}\, A.B).(y\ \&^{\{\pi x: A.B, \pi x: C.B\}} M).$$

[12] For example, if $A < B$ and $M{:}\, A$, then the redex $(\lambda x{:}\, B.x)M$ has type B but its reductum, M, has type A

All the overloaded functions will have their (static) type annotated on the &.

PRACTICAL EXCURSUS. *The use of annotations is needed only in the theoretic approach. They are needed because in an overloaded term $(\varepsilon \& M_1 \& M_2 \& \ldots \& M_n)$ the various subterms M_i may be different from λ-abstractions. In practice (that is, with multi-methods, generic functions, or the overloaded functors in Section 2.3 and Appendix A.2) this never happens. In all practical implementations of overloading, the composing functions are in λ-abstracted form. We think that $\lambda\Pi^{\&}$ won't make an exception. Thus, in a possible implementation inspired by this work, overloaded functions would be of the form $(\lambda x{:}\, A_1 . M_1 \& \lambda x{:}\, A_2 . M_2 \& \ldots \& \lambda x{:}\, A_n . M_n)$, which provides all it is needed in practice: each branch specifies the domain it was defined for, and its codomain is not strictly necessary to execution[13]. Thus, for overloaded functions of this form type annotations are unecessary.* □

3.4.4. Kinding

The deep interaction between overloading, subtyping, and late binding makes the language very powerful and expressive, but it complicates the kinding of overloaded types. In order to satisfy the subject reduction property not every set of π-types can be allowed in the language. Given an environment Γ, a well-kinded overloaded type $\{\pi x{:}\, A_i . B_i\}_{i \in I}$, besides being formed by well-kinded π-types, must satisfy three conditions:

(Normal types) For every $i \in I$ the type $\pi x{:}\, A_i . B_i$ is closed (it does not contain free *term* variables) and in normal form.

(Covariant types) For all $i, j \in I$ if $\Gamma \vdash A_i \leq A_j$ then $\Gamma, x{:}\, A_i \vdash B_i \leq B_j$

(Unique selection) For every type A whose free variables are in *Dom*(Γ) the set $\{A_i \mid \Gamma \vdash A \leq A_i, i \in I\}$ either is empty or has a unique least element.

Note that all these conditions, which define the kinding relation, are defined in terms of the subtyping relation. This is the reason why it is so important to have a subtyping relation whose definition does not directly depend on the kinding one (see Footnote 17).

Let us examine each condition in detail

- Suppose that the condition [normal types] was not fulfilled and open overloaded types were allowed in the calculus. Then we could write a term such as

$$M_1 \&^{\{\pi x:Ay.B, \pi x:AN.B\}} M_2$$

where A is a type family of the kind $\Pi x{:}A' . \star$ and N a term of the type A'.

Suppose that $y \notin Fv(M_1) \cup Fv(M_2) \cup Fv(N)$ and insert this term in a wider context

$$(\lambda y{:}S.(M_1 \&^{\{\pi x:Ay.B, \pi x:AN.B\}} M_2))N$$

Then after β-reduction we would obtain the term

$$M_1[y := N] \&^{\{\pi x:Ay.B, \pi x:AN.B\}[y:=N]} M_2[y := N]$$

[13] The absence of codomains could cause the subject-reduction property not to hold, but this would not affect the type-safety of the system.

that is

$$M_1 \&^{\{\pi x{:}AN.B,\pi x{:}AN.B\}} M_2$$

which clearly is not well-formed (more precisely, it is untypable) since branch selection is ambiguous.

A similar problem appears when the types are not in normal form[14].

- Condition [covariant types] ensures that during computation the type of a term may only decrease. More specifically, if we have a two-branched overloaded function M of type $\{\pi x{:}\, A_1.B_1, \pi x{:}\, A_2.B_2\}$ with $A_2 < A_1$ and we apply it to a term N that at compile-time has type A_1, then the compile-time type of $M{\bullet}N$ is B_1 (more precisely, $B_1[x := N]$). But if the normal form of N has type A_2 (which is possible, since $A_2 < A_1$) then the run-time type of $M{\bullet}N$ will be B_2 (more precisely, $B_2[x := N]$) and therefore $B_2 \leq B_1$ must hold (more precisely, it must hold under the hypothesis that $x : A_2$).
- Condition [unique selection] concerns the selection of the correct branch. Recall that if we apply an overloaded function of type $\{\pi x{:}\, A_i.B_i\}_{i\in I}$ to a term of type A, then the selected branch has type $\pi x{:}\, A_j.B_j$ such that $A_j = \min_{i\in I}\{A_i | A \leq A_i\}$. This condition is necessary and sufficient to ensure the existence and uniqueness of this branch.[15]

The last two conditions are already present in the $\lambda\&$-calculus where they have similar justifications.

The first condition instead is new and it resembles the meet-closure property of $F^{\&}_{\leq}$ [Cas96, Cas97]. Note however that this restriction is less constraining than meet-closure since it allows dependency on types of any form. Indeed, while this condition requires that in $\{\pi x{:}\, A_i.B_i\}_{i\in I}$ the various $\pi x{:}\, A_i.B_i$ must be closed, no restriction is imposed on the form of A_i and B_i which, therefore, may also be dependent or overloaded types (meet-closure requires the A_i's to be atomic). Thus there is a real, though limited, interaction between overloaded and dependent types.

PRACTICAL EXCURSUS. *The condition* [normal form] *is quite severe. From a practical point of view the requirement that types are in normal form is rather harmless. Instead, the condition that types are closed is very penalizing. The experience with object-oriented programming shows that overloaded functions are defined only at top level (that is, not in subterms, so that closure is trivially satisfied) and that in lower levels (in subterms) overloaded functions are used (applied) rather than defined (abstracted). We think that in many cases this should hold also for languages with dependent types (even though we have no evidence to support this claim). However, as a referee pointed us, this restriction rules out some interesting terms. For example, overloaded functions that return arrays parametrized by their length are not allowed, since their type would have open codomains (codomains with free variables different from the π-abstracted one):*

$$\pi n{:}\, \mathtt{nat}.\{\pi x{:}\, A.\mathtt{char[}n\mathtt{]}\ ,\ \pi x{:}\, B.\mathtt{int[}n\mathtt{]}\}$$

---

[14] Since β-reduction is a *compatible* reduction (see Footnote 4), then it can take place in every occurrence of a term and thus, in particular, in type annotations of overloaded terms. It is clear that normal forms would not be necessary if we did not reduce annotations.

[15] The restriction in [unique selection] that the free variables of A are in *Dom*(Γ), although natural, is quite technical and deep. It is crucial for proving that the satisfaction of [unique selection] is invariant under substitution. See the proof of Lemma 4.9 and note that it would not have worked if we had for example required the stronger condition $\Gamma \vdash A : \star$.

If n is not free in A and B, then such a situation can be dodged by swapping the arguments:

$$\{\pi x{:}\, A.\pi n{:}\, \mathtt{nat}.\mathtt{char}\,[n]\;,\; \pi x{:}\, B.\pi n{:}\, \mathtt{nat}.\mathtt{int}\,[n]\}$$

In any case we believe that in practice closure requirement for codomains could be relaxed. We would lose subject reduction but this should not affect type soundness (similarly to what happens for the system of [Ler94] we cited in the motivation section).

The closure of domains, instead is much more severe a restriction. For example, it does not allow one to write an overloaded function defined on arrays of different types since it would have a type with open domains, like this one

$$\pi n{:}\, \mathtt{nat}.\{\pi x{:}\, \mathtt{char}\,[n]\,.A\;,\; \pi x{:}\, \mathtt{int}\,[n]\,.B\} \tag{2}$$

In this case there is no simple expedient to satisfy closure. Nor we can easily relax the closure condition since while the type in (2) causes no harm to type soundness, a type such as

$$\pi n{:}\, \mathtt{nat}.\{\pi x{:}\, \mathtt{char}\,[n]\,.A\;,\; \pi x{:}\, \mathtt{char}\,[3]\,.B\} \tag{3}$$

must be forbidden since for $n = 3$ the two domains would be equated. The problem is how to weaken the closure requirement of [normal form] *so that the type in (2) is accepted and the one in (3) is rejected. This issue does not seem of immediate solution since one has to ensure some property for all possible term substitutions in domains. However, the point is well worth of studying and we look to do it in future work.* $\Box$

3.4.5. Typing

The typing system is obtained by adding the following three rules to the $\lambda\Pi_{\leq}$ typing rules:

T-ε
$$\frac{\Gamma \vdash \star}{\Gamma \vdash \varepsilon : \{\}}$$

T-&
$$\frac{\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i\leq n} \quad \Gamma \vdash N : \pi x{:}A_{n+1}.B_{n+1} \quad \Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star}{\Gamma \vdash M \&^{\{\pi x:A_i.B_i\}_{i\leq n+1}} N : \{\pi x{:}A_i.B_i\}_{i\leq n+1}}$$

T-OAPP
$$\frac{\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i\leq n} \qquad \Gamma \vdash N : A_j}{\Gamma \vdash M \bullet N : B_j[x := N]}$$

These typing rules deserve few comments. The first rule states that the empty overloaded function has an empty overloaded type. The second rule states that the type of an overloaded function is obtained by the union of the types of its branches, provided that the type resulting from this union is well-formed. The last rule states that if the argument of an overloaded function has type A_j then the j-th branch of the function *may* be selected.[16]

Note that in (T-&) the well-kindness of the resulting overloaded type (in particular the three conditions we just saw) must be checked.

3.4.6. Reduction

[16]Note that the branch effectively selected at run time may be different from the j-th branch either because A_j does not correspond to the "best approximating" branch, or because N is not a closed normal form (see next section).

The reduction for overloaded function applications in a context Γ is defined as follows:

If 1. N is closed and in normal form,
2. there exists $i \in [1..n]$ s.t. $\Gamma \vdash N\colon A_i$ and $\forall j \in [1..n]\ \Gamma \vdash N\colon A_j \Rightarrow \Gamma \vdash A_i \leq A_j$

then

$$(M_1 \&^{\{\pi x: A_h . B_h\}_{h=1..n}} M_2) \bullet N \rightarrow_{\beta\&} \begin{cases} M_1 \bullet N & \text{for } i < n \\ M_2 \cdot N & \text{for } i = n \end{cases}$$

The $\beta^{\&}$-reduction is simpler than what the definition above let suppose. The rewriting rule states that if we pass an argument N of type A_i to the overloaded function $(M_1 \&^{\{\pi x: A_h . B_h\}_{h=1..n}} M_2)$ then we select the branch defined for A_i (more precisely we select the branch M_2 if it is defined for A_i, otherwise the branch is searched in M_1). But in order to perform the reduction two preconditions must be fulfilled. The first condition requires that N is a closed normal form because, as explained in Section 3.4.2, we want to implement late-bound overloading. The second condition ensures that the most specific branch compatible with the type of the argument is selected. Indeed, imagine that an overloaded function with two branches, one for integers and the other for reals is applied to an argument of type integer. If integer is a subtype of real then by subsumption the argument has also type real. Thus either branch could be executed if the second condition would not ensure that the most specific one, namely the one for integers, is selected.[17]

Finally note that when the reduction is performed, all expressions that participate in the selection are closed. Thus, the definition of $\beta^{\&}$ does not depend on the typing assignments in Γ but just only on its subtyping declarations.[18]

TECHNICAL EXCURSUS. *The requirement that N is a closed normal form is very strong. As explained for $\lambda\&$ in Section 7.2 of [Cas97] this condition may be weakened. For example, one can always safely perform the reduction when the involved overloaded function has only one branch, or when the type of the argument is a leaf of the type hierarchy. An interesting choice (but others are possible) is to weaken the $\beta^{\&}$ rule as follows:*

For A_i such that $\Gamma \vdash N : A_i$ and $\forall j \in [1..n]\ \Gamma \vdash N : A_j \Rightarrow \Gamma \vdash A_i \leq A_j$,
If N is closed and in normal form **or** $\{A_j \mid 1 \leq j \leq n, \Gamma \vdash A_j \leq A_i\} = \{A_i\}$, *then*

$$(M_1 \&^{\{\pi x: A_h . B_h\}_{h=1..n}} M_2) \bullet N \rightarrow_{\beta\&} \begin{cases} M_1 \bullet N & \textit{for } i < n \\ M_2 \cdot N & \textit{for } i = n \end{cases}$$

In words, when selecting a branch we check whether there are other branches with smaller domain. If not, we know that the selection cannot further change and therefore we perform the reduction even if N is not closed.

---

[17] The $\beta^{\&}$-reduction depends on typing, which in turn depends on kinding. The subtyping relation depends on β-conversion (rule S-ApR), therefore, strictly speaking, the subtyping relation is not independent from the typing and kinding relations. So it seems that by defining the $\beta^{\&}$ reduction we reintroduced the circularity we so hardly struggled against in Section 3.3.2. However, this dependence is far milder than the one of $\lambda P_{\leq}$ and it does not cause any problem in the meta-theoretic study. For example, in the proof for substitution of subtyping (Lemma 4.6, Case S-ApR), the induction concerns only the subtyping derivation, but not the typing or kinding judgments.

[18] This greatly simplifies the treatment of reduction. The definition of reduction can be given for a generic context Γ but it has not to deal with it. So all the proofs of this article that deal with $\beta^{\&}$ are given for a generic Γ. It would be quite different if the reduction depended on the type assignments in Γ. In that case we would have to define rules that handle contexts, such as:

$$\frac{\Gamma, (x\colon A) \vdash M_1 \rightarrow_\beta M_2}{\Gamma \vdash \lambda x\colon A.M_1 \rightarrow_\beta \lambda x\colon A.M_2}$$

and deal with them explicitly in the proofs.

This rule is interesting because, for example, it allows to deduce under the context $n\colon \mathtt{int}$ *that:*[19]

$$((\lambda x\colon \mathtt{int}.x+x)\&(\lambda x\colon \mathtt{string}.x@x))\bullet n \quad \twoheadrightarrow \quad n+n$$

However, such a rule would complicate the calculus (for example, this same example shows that in this case reductions depend on the type assignments of the context Γ*, with the problems described in Footnote 18). Furthermore, this modification would not be straightforward (for example, it is not clear how to prove that some properties are preserved under substitution, such as the property of minimality in a set of open types). Therefore, we prefer to proceed in this work as for* $\lambda\&$ *and consider just the simpler formulation.* $\square$

3.4.7. Examples

As a first example of use of overloaded dependent types we can think of generalizing the function `string_to_array` defined in the Introduction so that it can be applied to natural numbers as well. Since a natural number n needs $\lceil \log_{10} n \rceil$ digits to be represented, then such a function will have the following type:

$$\{\ \pi x\colon \mathtt{nat}.\mathtt{char}[\lceil \log_{10} x \rceil]\ ,\ \pi x\colon \mathtt{string}.\mathtt{char}[\mathit{length}(x)]\ \}$$

To give a more detailed example we show two distinct encodings of the cartesian product. The first one requires indexed types, is more complicated, but also more efficient. The second one is simpler, works for all types, but since it uses late binding, it requires run-time type inference.

Let ι be an atomic type with two constants[20] $1\colon\iota$ and $2\colon\iota$, and A a type indexed over ι. This can be expressed by the following context $\Gamma_0 \equiv \iota\colon\star\ ,\ 1\colon\iota\ ,\ 2\colon\iota\ ,\ A:\Pi x\colon\iota.\star$. Add a further constant $\mathtt{?} : \pi x\colon\iota.A1 \to A2 \to Ax$ (note that $A1$, $A2$, Ax, $x1$, etc. stand for the applications $A(1)$, $A(2)$, $A(x)$ and $x(1)$) with the following semantics:

$$\mathtt{?}xa_1a_2 = \begin{cases} a_1 & \text{for } x = 1 \\ a_2 & \text{otherwise} \end{cases}$$

(the last example of this section shows how to encode $\mathtt{?}$). Then, we can use dependent types to define the cartesian products of $A1$ and $A2$:

$$\begin{aligned} A1\times A2 &= \pi x\colon\iota.Ax \\ (\_,\_) &= \lambda a_1\colon A1.\lambda a_2\colon A2.\lambda x\colon\iota.\mathtt{?}xa_1a_2 \\ \text{fst} &= \lambda x\colon A1\times A2.x1 \\ \text{snd} &= \lambda x\colon A1\times A2.x2 \end{aligned}$$

A simpler encoding can be obtained by using overloaded types together with two atomic types α_1 and α_2 and two constants $p_1\colon\alpha_1$ and $p_2\colon\alpha_2$ (in this and in the following example we omit type annotations and ε's):

$$\begin{aligned} A1\otimes A2 &= \{\pi x\colon\alpha_1.A1\ ,\pi x\colon\alpha_2.A2\} \\ (\_,\_) &= \lambda a_1\colon A1.\lambda a_2\colon A2.(\lambda x\colon\alpha_1.a_1 \ \&\ \lambda x\colon\alpha_2.a_2) \\ \text{fst} &= \lambda x\colon A1\otimes A2.x\bullet p_1 \\ \text{snd} &= \lambda x\colon A1\otimes A2.x\bullet p_2 \end{aligned}$$

---

[19] We use the operator @ to denote string concatenation. This example was suggested by one of the referees.

[20] We did not explicitly consider constants in the formal syntax of $\lambda\Pi^{\&}$. Rather, we take the attitude of [Bar92] and call constants all variables that we "engage" not to abstract.

Then it is possible to define generic *first* and *second* operators that work with both encodings. For example *first* can be defined as

$$(\lambda x\colon (A1\times A2).x1 \ \&\ \lambda x\colon (A1\otimes A2).x\bullet p_1)$$

whose type is

$$\{\pi x\colon (\pi y\colon \iota.Ay).A1\ ,\ \pi x\colon \{\pi y\colon \alpha_1.A1\,,\pi y\colon \alpha_2.A2\}.A1\}$$

Note that all these definitions can be applied to pairs of terms whose types are subtypes of $A1$ and $A2$.

As a last example, imagine that we want to use dependent types to encode triples. We want to define the encoding so that we can use triples where pairs are expected (as if they were record types with labels 1, 2, and 3). This can be obtained by concatenating the two following contexts:
$\Gamma_0 \equiv \iota_{123}\colon\star\ ,\ \iota_3 \leq \iota_{123}\colon\star\ ,\ \iota_{12} \leq \iota_{123}\colon\star\ ,\ \iota_2 \leq \iota_{12}\colon\star\ ,\ \iota_1 \leq \iota_{12}\colon\star\ ,\ 1\colon\iota_1\ ,\ 2\colon\iota_2\ ,\ 3\colon\iota_3$
$\Gamma_1 \equiv A\colon\Pi x\colon\iota_{123}.\star\ ,\ -\colon\pi x\colon\iota_{123}.Ax$
Context Γ_0 declares three singleton types ι_1, ι_2, ι_3 respectively containing constants 1, 2, and 3. It also declares two unions of these singletons, ι_{12} (that contains both ι_1 and ι_2) and ι_{123} (that contains all the other types).[21] Context Γ_1 declares the type A indexed over ι_{123}, together with a constant $-$ such that $-i : Ai$ for $i = 1, 2, 3$. Finally we encode ? as follows:

$$? = \lambda x\colon\iota_{123}.\lambda a_1\colon A1.\lambda a_2\colon A2.\lambda a_3\colon A3.(\lambda y\colon\iota_1.a_1\&\lambda y\colon\iota_2.a_2\&\lambda y\colon\iota_3.a_3\&\lambda y\colon\iota_{123}.-y)\bullet x$$

whose type[22] is $\pi x\colon\iota_{123}.A1 \to A2 \to A3 \to Ax$ and whose semantics clearly is

$$?xa_1a_2a_3 = \begin{cases} a_1 & \text{for } x = 1\\ a_2 & \text{for } x = 2\\ a_3 & \text{for } x = 3\end{cases}$$

With these declarations the dependent-types-based encoding for pairs is as before. Just the pairing operator has to be modified to take into account the fourth argument of ?:

$$\begin{aligned} A1\times A2 &= \pi x\colon\iota_{12}.Ax\\ (\_,\_) &= \lambda a_1\colon A1.\lambda a_2\colon A2.\lambda x\colon\iota_{12}.?xa_1a_2(-3)\\ \text{fst} &= \lambda x\colon A1\times A2.x1\\ \text{snd} &= \lambda x\colon A1\times A2.x2\end{aligned}$$

Triples are similar

$$\begin{aligned} A1\times A2\times A3 &= \pi x\colon\iota_{123}.Ax\\ (\_,\_,\_) &= \lambda a_1\colon A1.\lambda a_2\colon A2.\lambda a_3\colon A3.\lambda x\colon\iota_{123}.?xa_1a_2a_3\end{aligned}$$

[21] Instead of $2\colon\iota_2, 3\colon\iota_3$ we could equivalently have declared $s\colon\{\iota_1 \to \iota_2\ ,\ \iota_2 \to \iota_3\}$, so to have $2 \equiv s\bullet 1$ and $3 \equiv s\bullet(s\bullet 1)$

[22] Note that, strictly speaking, the type of the inner overloaded function is not well-kinded: since $\iota_i \leq \iota_{123}$, then the [covariant types] condition requires $y\colon\iota_i \vdash Ai \leq Ay$. This semantically holds: ι_i is a singleton containing i, thus $y\colon\iota_i$ implies $y = i$. But this equality can be proved only by extending the system with the distinguished singleton types introduced by David Aspinall in [Asp95]. This would lead us too far.

Note that by the rule S-π $A1\times A2\times A3 \leq A1\times A2$. Thus, thanks to subtyping, we need not to define first and second projections for triples since the functions fst and snd defined for pairs work also for triples (in object-oriented terminology we would say that triples "inherit" first and second projections from pairs). Instead, we have to define the third projection. A term may be statically typed as a pair and dynamically become a triple. Thus, it is interesting to define the third projection also for pairs. If the pair dynamically becomes a triple then this projection function returns the third component of the triple, otherwise it returns -3. This is obtained by the following overloaded function:

$$\text{trd} = (\lambda y{:}\,A1\times A2.{-3})\ \&\ (\lambda y{:}\,A1\times A2\times A3.y3)$$

whose type is $\{\pi y{:}(\pi x{:}\,i_{12}.Ax).A3\ ,\ \pi y{:}(\pi x{:}\,i_{123}.Ax).A3\}$.

3.4.8. Properties

The hardest and most technical part of this work is to prove that $\lambda\Pi^{\&}$ enjoys good theoretical properties. This is what the rest of this article is devoted to. More precisely, after having formally presented $\lambda\Pi^{\&}$ (Section 4.1) we proceed as follows.

First, we prove the confluence of the calculus. Namely, let $\beta = \beta_1 \cup \beta_2 \cup \beta^{\&}$; if $M \twoheadrightarrow_\beta M_1$ and $M \twoheadrightarrow_\beta M_2$ then there exists N such that $M_1 \twoheadrightarrow_\beta N$ and $M_2 \twoheadrightarrow_\beta N$ (Section 4.2).

Second, since we started from a set of rules, those of $\lambda\Pi_{\leq}$, that do not include (general) transitivity and reflexivity rules we have to prove that the subtyping relation on types is a preorder, that is, that the two rules are admissible (Sections 4.3 and 4.4).

Third, we prove that our type system is sound since well-typed terms rewrite only into well-typed terms. That is, if $\Gamma \vdash M : A$ and $M \twoheadrightarrow N$ then $\Gamma \vdash N : A$ (Section 4.5).

Thanks to the absence of transitivity from the subtyping rules, it is then not very difficult to prove that $\lambda\Pi^{\&}$ is a conservative extension of both $\lambda\Pi_{\leq}$ and $\lambda\&$ (Section 4.6).

A delicate point is that, since $\lambda\Pi^{\&}$ extends $\lambda\&$, it inherits from the latter non termination: in $\lambda\&$ it is possible to encode a fix-point combinator of type $(A \to A) \to A$ for every type A (see §6.2 of [Cas97]), and this same technique applies to $\lambda\Pi^{\&}$ as well. However in $\lambda\&$ it is possible to single out an interesting class of sub-calculi that are strongly normalizing. We show that this result extends to $\lambda\Pi^{\&}$ (Section 5). The interest in normalization properties in $\lambda\Pi^{\&}$ stems from the fact that subtyping relies on β-conversion. Terms may appear in types and β-conversion of terms is used to define subtyping. Strong normalization of terms implies decidability of β-equality, which implies decidability of subtyping. Indeed, in presence of the decidability of β-conversion it is easy to lift the proof of the decidability of the subtyping (and thus of typing) relation of $\lambda\Pi_{\leq}$ (this proof is given in Section 6.4) to the strongly normalizing subsystems of $\lambda\Pi^{\&}$. Thus, in these subsystems of $\lambda\Pi^{\&}$ we have decidability for both subtyping and typing relations (which is the main result of Section 7, and does not hold for the whole $\lambda\Pi^{\&}$).

In Section 6 we prove some properties that are specific to $\lambda\Pi_{\leq}$, namely that the subtyping rules of $\lambda\Pi_{\leq}$ describe an algorithm, that $\lambda\Pi_{\leq}$ is equivalent to $\lambda P_{\leq}$, and the already cited proof of decidability of the subtyping relation of $\lambda\Pi_{\leq}$ that is then used in the last section for studying decidability in $\lambda\Pi^{\&}$ (Section 7).

4. FORMAL PRESENTATION OF $\lambda\Pi^{\&}$

In this section we give the formal definition of $\lambda\Pi^{\&}$ and we prove that it enjoys several fundamental properties.

4.1. The system $\lambda\Pi^{\&}$

The system $\lambda\Pi^{\&}$ is an extension of $\lambda\&$ with first order types. There are four syntactic categories: contexts, denoted by Γ, kinds, denoted by K, types, denoted by A, B, C, D, and terms, denoted by M, N, P and Q. We use U to range over the last three syntactic categories (all these meta-variable may appear indexed). Sometimes we will denote an overloaded type as a set of π-types indexed over a set of indexes (typically, they will be initial segments of natural numbers); we use I and J to range over set of indexes and h, i, j, k to range over indexes. There are four judgment forms on these expressions:

$\Gamma \vdash K$	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

These judgments are same as those in $\lambda\Pi_{\leq}$.

Pre-terms, pre-types, pre-kinds, and pre-contexts (i.e., possibly not well-formed terms, types, kinds, and contexts) are those in $\lambda\Pi_{\leq}$ extended by an empty overloaded function, ε, non empty overloaded functions $M\&^{A}M$, applications of overloaded functions $M \bullet M$, and overloaded pre-types, $\{\pi x{:}A_i.B_i\}_{i\in I}$,. The structure of kinds and contexts is unchanged.

$$
\begin{array}{lcl}
M & ::= & x \mid \lambda x{:}A.M \mid MM \mid \varepsilon \mid\ M\&^{A}M \mid M \bullet M \\
A & ::= & \alpha \mid \pi x{:}A.A \mid \Lambda x{:}A.A \mid AM \mid \{\pi x{:}A.A, ..., \pi x{:}A.A\} \\
K & ::= & \star \mid \Pi x{:}A.K \\
\Gamma & ::= & <>\ \mid \Gamma, x : A \mid \Gamma, \alpha : K \mid \Gamma, \alpha \leq A : K
\end{array}
$$

4.1.1. Rules

The set of rules defining $\lambda\Pi^{\&}$ is obtained by adding to the rules of $\lambda\Pi_{\leq}$ the kinding and subtyping rules for overloaded types and the typing rules for the three new terms for overloading. The complete set of rules is given in Appendix A.1.

The subtyping rule for overloaded types is

$$
\text{S-OVER}\qquad \frac{\forall j \in J\ \exists i \in I \qquad \Gamma \vdash \pi x{:}A_i.B_i \leq \pi y{:}C_j.D_j}{\Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\in I} \leq \{\pi y{:}C_j.D_j\}_{j\in J}}
$$

while the kinding rule is:

$$
\text{K-OVER}\quad \frac{\begin{array}{c}
\Gamma \vdash \star \qquad \forall i \in I\ .\ \Gamma \vdash \pi x{:}A_i.B_i : \star \\
\forall i \in I\ .\ \pi x{:}A_i.B_i \text{ is closed and in normal form} \\
\forall i, j \in I\ .\ \Gamma \vdash A_i \leq A_j \Rightarrow \Gamma, x : A_i \vdash B_i \leq B_j \\
\forall A.Fv(A) \subseteq Dom(\Gamma) \Rightarrow\ ((\forall i \in I.\ \Gamma \not\vdash A \leq A_i)\vee \\
\qquad (\exists! i \in I.\ \Gamma \vdash A \leq A_i \wedge \forall j \in I\ \Gamma \vdash A \leq A_j \Rightarrow \Gamma \vdash A_i \leq A_j))
\end{array}}{\Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\in I} : \star}
$$

This huge rule simply formalizes the conditions that we stated in Section 3.4.4. Namely, the first line in the premises states that a well-kinded overloaded type is formed by well-kinded

π-types ($\Gamma \vdash \star$ ensures that Γ is well-formed even for an empty I). These types, according to the second line, are closed and in normal form. The third line requires covariance while the last two lines formalize the condition of unique selection.

Finally we add to the typing rules of $\lambda\Pi_{\leq}$ the following rules:

$$\text{T-}\varepsilon \qquad \frac{\Gamma \vdash \star}{\Gamma \vdash \varepsilon : \{\}}$$

$$\text{T-\&} \quad \frac{\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i \leq n} \quad \Gamma \vdash N : \pi x{:}A_{n+1}.B_{n+1} \quad \Gamma \vdash \{\pi x{:}A_i.B_i\}_{i \leq n+1} : \star}{\Gamma \vdash M \&^{\{\pi x:A_i.B_i\}_{i \leq n+1}} N : \{\pi x{:}A_i.B_i\}_{i \leq n+1}}$$

$$\text{T-OAPP} \qquad \frac{\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i \leq n} \qquad \Gamma \vdash N : A_j}{\Gamma \vdash M \bullet N : B_j[x := N]}$$

4.1.2. Notions of reduction

The β-reduction is defined as the *compatible closure* of the union of three *notions of reduction* (for definitions see §3.1 of [Bar84]), β_1, β_2, and $\beta^{\&}$ defined in Sections 3.2 and 3.4.6.

In the following we use $\leftarrow_R$ to denote the symmetric relation of $\rightarrow_R$, and use $\twoheadrightarrow_R$ (respectively $=_R$) to denote reflexive, transitive closure of the reduction $\rightarrow_R$ (respectively of $\rightarrow_R \cup \leftarrow_R$). We use U^R to denote the R-normal-form of U and $\equiv$ to denote syntactic identity of expressions.

We write $\Gamma \vdash J$ to denote an arbitrary judgment and with an abuse of notation we will write $J \rightarrow_\beta J'$ to denote that J' is obtained by replacing a β-redex in J by its reductum.

4.2. Confluence

The first property that we prove for $\lambda\Pi^{\&}$ is confluence (expressions U in this section are not assumed to be well-typed. We just require that all the type annotations of overloaded terms occurring in them are closed, so that substitutions do not affect them). The proof is a simple application of the Hindley-Rosen Lemma [Hin64, Ros73].

LEMMA 4.1 (Hindley-Rosen Lemma). *Let R_1, R_2 be two notions of reduction. If R_1, R_2 are confluent and $\twoheadrightarrow_{R_1}$ commutes with $\twoheadrightarrow_{R_2}$ then $R_1 \cup R_2$ is confluent.*

Set now $R_1 = \beta_1 \cup \beta_2$ and $R_2 = \beta^{\&}$. If we prove that these notions of reduction satisfy the hypothesis of the lemma above, we obtain confluence of our system. The confluence of $\beta_1 \cup \beta_2$ is easy to prove since it essentially reduces to the confluence of $\lambda\Pi$. For $\beta^{\&}$ it is easy to verify that it satisfies the diamond property: for example consider the two $\beta^{\&}$-reductions,

$$M_2 N \ \leftarrow_{\beta^{\&}} \ (M_1 \&^A M_2) \bullet N \ \rightarrow_{\beta^{\&}} \ (M_1 \&^A M_2') \bullet N$$

where $M_2 \ \rightarrow_{\beta^{\&}} \ M_2'$. A has not been changed in the second reduction, and the branch selection in $(M_1 \&^A M_2') \bullet N$ is determined by A and the typing of N, so this expression $\beta^{\&}$-reduces to $M_2' N$. On the other hand, $M_2 N \rightarrow_{\beta^{\&}} M_2' N$.

Thus it remains to prove that the two notions of reduction commute. To that end we can use the Lemma 3.3.6 of [Bar84], which is restated here. For any reduction notion R, let $\overset{\equiv}{\rightarrow}_R$ denote the reflexive closure of R.

LEMMA 4.2 ([Bar84]). *Let R_1, R_2 be two notions of reduction, U be a term. If $N_1 \leftarrow_{R_2} U \rightarrow_{R_1} N_2$, then $\exists N,\ N_1 \twoheadrightarrow_{R_1} N \stackrel{=}{\leftarrow}_{R_2} N_2$. then $\twoheadrightarrow_{R_1}, \twoheadrightarrow_{R_2}$ commute.*

Now consider Lemma 4.2. The next proposition shows that for $R_1 = \beta^{\&}, R_2 = \beta_1 \cup \beta_2$ the hypothesis of the lemma are satisfied. Therefore we can conclude that $\twoheadrightarrow_{\beta^{\&}}$ commutes with $\twoheadrightarrow_{\beta_1 \cup \beta_2}$.

PROPOSITION 4.3 (Weak commutativity). *If $N_1 \leftarrow_{\beta_1 \cup \beta_2} U \rightarrow_{\beta^{\&}} N_2$, then $\exists N_3\ s.t.\ N_1 \twoheadrightarrow_{\beta^{\&}} N_3 \leftarrow^{=}_{\beta_1 \cup \beta_2} N_2$*

Confluence of β-reduction follows from the Hindley-Rosen Lemma:

COROLLARY 4.4 (Confluence). *Suppose U, U', U'' are pre-expressions. If $U \twoheadrightarrow_\beta U'$ and $U \twoheadrightarrow_\beta U''$, then there exists a pre-expression V such that $U' \twoheadrightarrow_\beta V$ and $U'' \twoheadrightarrow_\beta V$.*

4.3. Structural Properties and β_2-reduction

In this section we prove some structural properties of the $\lambda\Pi^{\&}$ typing and subtyping systems, as well as some properties of the β_2-reduction. All these properties are rather technical and not very interesting by their own sake. But they are necessary to the proofs of Section 4.4. Thus we strongly suggest to the reader to skip this section in the first reading and to pass directly to Section 4.4.

In all this Section 4.3 long, J denotes either a typing ($M\colon A$), or a kinding ($A\colon K$), or a context formation (K) judgment (i.e., subtyping judgments are not included).

4.3.1. Substitution

Since subtyping system is independent from typing and kinding and overloaded types are formed only by closed types, then it is not very difficult to prove substitution property for subtyping, that is if $\Gamma \vdash B \leq C$ is derivable, then for any term M, the judgment $\Gamma[x := M] \vdash B[x := M] \leq C[x := M]$ is derivable as well.

But first let us precisely define substitution for a context.

DEFINITION 4.5 (Substitution of context). *The substitution $\Gamma[x := M]$ of a variable x by a term M in the context Γ is defined as follows:*

$$\begin{array}{lll} <>[x := M] & =_{def} & <> \\ (\Gamma_1, y : A)[x := M] & =_{def} & \Gamma_1[x := M], y : A[x := M] \\ (\Gamma_1, x : A)[x := M] & =_{def} & \Gamma_1[x := M] \end{array}$$

Note that, if Γ is a well-formed context (i.e., $\Gamma \vdash \star$) and $\Gamma = \Gamma_1, x : C, \Gamma_2$, then $\Gamma[x := M] = \Gamma_1, \Gamma_2[x := M]$ since $x \notin \Gamma_1$.

LEMMA 4.6 (Substitution for subtyping). *If $\Gamma \vdash B \leq C$, then $\Gamma[x := M] \vdash B[x := M] \leq C[x := M]$. Furthermore the depth of the derivation of $\Gamma[x := M] \vdash B[x := M] \leq C[x := M]$ is not greater than the depth of the derivation of $\Gamma \vdash B \leq C$*

Proof. By induction on the depth of the derivation of $\Gamma \vdash B \leq C$ and performing a case analysis on the last rule of the derivation. $\Box$

Note that in the lemma above there is no requirement on M (e.g. it may be non-typable).

If the substitution variable does not appear in the subtyping judgment, then the converse of the above result holds:

LEMMA 4.7. $x \notin Fv(A) \cup Fv(B) \ \wedge \ \Gamma[x := M] \vdash A \leq B \ \Rightarrow \ \Gamma \ \vdash A \leq B$

Proof. Straightforward induction on the depth of the derivation of $\Gamma \vdash A \leq B$. $\Box$

Next we study the preservation under substitution of the conditions of well-formation of overloaded types. First,we consider the covariance condition:

LEMMA 4.8 (Preservation of covariance by substitution). *If* $x \notin Fv(A) \cup Fv(A') \cup Fv(B) \cup Fv(B')$ *and* $\Gamma \vdash A \leq A' \Rightarrow \Gamma \vdash B \leq B'$*, then* $\Gamma[x := M] \vdash A \leq A' \Rightarrow \Gamma[x := M] \vdash B \leq B'$

Proof.

$$\begin{array}{llll} \Gamma[x := M] \vdash A \leq A' & \Rightarrow & \Gamma \vdash A \leq A' & \text{Lemma 4.7} \\ & \Rightarrow & \Gamma \vdash B \leq B' & \text{By assumption} \\ & \Rightarrow & \Gamma[x := M] \vdash B \leq B' & \text{Lemma 4.6} \end{array}$$

$\Box$

Then we consider the uniqueness of selection (see also Footnote 15):

LEMMA 4.9 (Preservation of uniqueness by substitution).
Let $\{A_i \mid i \in I\}$ *be a set of closed types. Then*

$$\begin{array}{l} \forall A \ s.t. \ Fv(A) \subseteq Dom(\Gamma), \\ \qquad ((\forall i \in I. \, \Gamma \not\vdash A \leq A_i) \vee \\ \qquad\qquad (\exists! i \in I. \, \Gamma \vdash A \leq A_i \ \wedge \ \forall j \in I \ \Gamma \vdash A \leq A_j \Rightarrow \Gamma \vdash A_i \leq A_j)) \end{array} \tag{4}$$

implies

$$\begin{array}{l} \forall A \ s.t. \ Fv(A) \subseteq Dom(\Gamma[x := M]) \\ \qquad ((\forall i \in I. \, \Gamma[x := M] \not\vdash A \leq A_i) \vee \\ \qquad\qquad (\exists! i \in I. \, \Gamma[x := M] \vdash A \leq A_i \ \wedge \\ \qquad\qquad\qquad \forall j \in I \ \Gamma[x := M] \vdash A \leq A_j \Rightarrow \Gamma[x := M] \vdash A_i \leq A_j)) \end{array}$$

Proof. Fix an A such that $Fv(A) \subseteq Dom(\Gamma[x := M])$. Note that $Fv(A) \subseteq Dom(\Gamma[x := M]) \subseteq Dom(\Gamma)$. Therefore the equation (4) holds for this particular A.

If the first clause of (4) holds, that is $(\forall i \in I. \, \Gamma \not\vdash A \leq A_i)$, then $(\forall i \in I. \, \Gamma[x := M] \not\vdash A \leq A_i)$ holds. Indeed, imagine that there exists $h \in I$ such that $\Gamma[x := M] \vdash A \leq A_h$. Since $Fv(A) \subseteq Dom(\Gamma[x := M])$ then $x \notin Fv(A)$. Therefore we can apply Lemma 4.7 and obtain $\Gamma \vdash A \leq A_h$. Contradiction.

If the second clause of (4) holds, then let h be the unique index in I such that

$$\Gamma \vdash A \leq A_h \ \wedge \ \forall j \in I \ \Gamma \vdash A \leq A_j \Rightarrow \Gamma \vdash A_h \leq A_j \tag{5}$$

We first prove the existence part by showing that

$$\Gamma[x := M] \vdash A \leq A_h \ \wedge \ \forall j \in I \ \Gamma[x := M] \vdash A \leq A_j \Rightarrow \Gamma[x := M] \vdash A_h \leq A_j \tag{6}$$

Observe that

$$\begin{array}{lll} \Gamma \vdash A \leq A_h & & \text{Formula (4)} \\ \qquad \Rightarrow & \Gamma[x := M] \vdash A[x := M] \leq A_h[x := M] & \text{Lemma 4.6} \\ \qquad \Rightarrow & \Gamma[x := M] \vdash A \leq A_h & x \notin Fv(A) \text{ and } A_h \text{ is closed} \end{array}$$

and that

$$\begin{array}{lll}
\Gamma[x := M] \vdash A \leq A_j & & \\
& \Rightarrow\ \Gamma \vdash A \leq A_j & \text{Lemma 4.7} \\
& \Rightarrow\ \Gamma \vdash A_h \leq A_j & \text{Implication (5)} \\
& \Rightarrow\ \Gamma[x := M] \vdash A_h[x := M] \leq A_j[x := M] & \text{Lemma 4.6} \\
& \Rightarrow\ \Gamma[x := M] \vdash A_h \leq A_j & A_h, A_j \text{ are closed}
\end{array}$$

These two last observations imply (6).

In order to prove the uniqueness of h for (6), assume that there exists $k \in I$ that satisfies (6). This implies that $\Gamma \vdash A \leq A_k$ (Lemma 4.7) and that for all $j \in I$

$$\begin{array}{lll}
\Gamma \vdash A \leq A_j & & \\
& \Rightarrow\ \Gamma[x := M] \vdash A \leq A_j & \text{Lemma 4.6} \\
& \Rightarrow\ \Gamma[x := M] \vdash A_k \leq A_j & \text{Assumption for (6)} \\
& \Rightarrow\ \Gamma \vdash A_k \leq A_j & \text{Lemma 4.7} \\
& \Rightarrow\ h = k & \text{By the uniqueness of } h \text{ for (5)}
\end{array}$$

□

PROPOSITION 4.10 (Context properties).
1. If $\Gamma \vdash J$ is provable, then for every prefix Γ' of Γ, $\Gamma' \vdash \star$ is provable by a derivation of strictly lesser depth.

2. $\Gamma_1, x : A, \Gamma_2 \vdash J \;\Rightarrow\; \Gamma_1 \vdash A : \star$, where $\Gamma_1 \vdash A : \star$ has a smaller proof than $\Gamma_1, x : A, \Gamma_2 \vdash J$.

3. If $\Gamma, \Gamma' \vdash J$ is provable and $\Gamma, x : A, \Gamma' \vdash \star$ then also $\Gamma, x : A, \Gamma' \vdash J$ is provable (weakening).

Proof. The first and third points can be easily proved by induction on the depth of the derivation of the judgment $\Gamma \vdash J$. The second point is a straightforward consequence of the first one. □

PROPOSITION 4.11 (Substitution). *Let $\Gamma \equiv \Gamma_1, x{:}C, \Gamma_2$. If $\Gamma \vdash J$ and $\Gamma_1 \vdash M{:}C$ are derivable, then also $\Gamma[x := M] \vdash J[x := M]$ is derivable. More precisely:*

1. $\Gamma \vdash K \;\Rightarrow\; \Gamma[x := M] \vdash K[x := M]$
2. $\Gamma \vdash A{:}K \;\Rightarrow\; \Gamma[x := M] \vdash A[x := M] : K[x := M]$
3. $\Gamma \vdash N{:}A \;\Rightarrow\; \Gamma[x := M] \vdash N[x := M] : A[x := M]$

(Substitution for subtyping has already been proved in Lemma 4.6)

Proof. By induction on the depth of the the derivation of $\Gamma \vdash J$, by a case analysis on the last applied rule. The proof is quite straightforward. We just hint the following points. Proposition 4.10 must be used for the case F-TERM when the variable introduced by the rule is x. The case for T-VAR is the only case which uses the hypothesis $\Gamma \vdash M : C$. The case T-SUB is proved by applying Lemma 4.6. The case K-CONV uses the property that $K =_\beta K'$ implies $K[x := M] =_\beta K'[x := M]$. The cases for the elimination rules (i.e., K-APP, T-APP, T-OAPP) hold because of the property $U[x := M][y := N[x := M]] = U[y := N][x := M]$. The case K-OVER uses Lemmas 4.8 and 4.9. □

4.3.2. Kinding Properties

LEMMA 4.12. *If* $\Gamma \vdash \star$ *and* $x \in Dom(\Gamma)$ *then* $\Gamma \vdash \Gamma(x) : \star$

Proof. A simple induction on the length of Γ. □

A simple but important consequence of the previous result is

PROPOSITION 4.13. *If* $\Gamma \vdash M : A$ *then* $\Gamma \vdash A : \star$.

Proof. By induction on the depth of the derivation of $\Gamma \vdash M : A$ and performing a case analysis on the last applied rule. Use Lemma 4.12 in the case T-VAR and the second point of Proposition 4.11 for the cases T-APP and T-OAPP. □

4.3.3. Generation Principle

The generation for kinding tells us what information we can infer from a kinding judgment about a kind.

PROPOSITION 4.14 (Generation for kinding).
$\Gamma \vdash \alpha : K \quad \Rightarrow \quad K =_\beta Kind_\Gamma(\alpha)$
$\Gamma \vdash \pi x{:}A.B : K \quad \Rightarrow \quad K \equiv \star \ \wedge\ \Gamma, x : A \vdash B : \star$
$\Gamma \vdash \Lambda x{:}A.B : K \quad \Rightarrow \quad \exists K' \ s.t.\ K =_\beta \Pi x{:}A.K' \ \wedge\ \Gamma, x : A \vdash B : K'$
$\Gamma \vdash AM : K \quad \Rightarrow \quad \exists B, K' \ s.t.\ \Gamma \vdash A : \Pi x{:}B.K' \ \wedge\ \Gamma \vdash M : B \wedge\ K'[x := M] =_\beta K$
$\Gamma \vdash \{\pi x{:}A_i.B_i\}_{i \in I} : K \quad \Rightarrow \quad K \equiv \star \ \wedge\ \forall i \in I,\ \Gamma, x : A_i \vdash B_i : \star$

Proof. By inspection of the kinding rules. □

PROPOSITION 4.15 (Uniqueness of kinds). *If* $\Gamma \vdash A : K$ *and* $\Gamma \vdash A : K'$, *then* $K =_\beta K'$

Proof. Uniqueness of kinding can be obtained by observing the fact that all kinds are of the form $\Pi x_1{:}A_1 \ ..\ \Pi x_n{:}A_n.\star$ and $\Pi x_1{:}A_1 \ ..\ \Pi x_n{:}A_n.\star =_\beta \Pi x_1{:}A'_1 \ ..\ \Pi x_m{:}A'_m.\star$ iff $n = m, A_i =_\beta A'_i, i = 1..n$. □

4.3.4. Context change

The properties in this subsection concern the preservation of judgment derivability with respect to change of context.

PROPOSITION 4.16 (Bound change for subtyping). *If* $\Gamma_1, x : A, \Gamma_2 \vdash B \leq C$, *then* $\Gamma_1, x : A', \Gamma_2 \vdash B \leq C$

Proof. By induction on the derivation of $\Gamma_1, x : A, \Gamma_2 \vdash B \leq C$. □

This property shows that subtyping does not depend on the types of the context term variables. The only declarations in Γ that concerns subtyping are bounded kind assignments such as $\alpha \leq A : K$. [23]

Now, we study the preservation of the conditions in the overloaded type formation under type changes of term variables in the context. First, we show the preservation of covariance condition under term bound change.

[23] Of course this holds in our system only because the definition of $\leq$ does not depend on kinding judgments. If instead of the subtyping relation we had used a kinding relation this would not hold (e.g. see Proposition 4.19).

LEMMA 4.17 (Preservation of covariance by term bound change). *If* $\Gamma_1, x : C, \Gamma_2 \vdash A \leq A'$ *implies* $\Gamma_1, x : C, \Gamma_2 \vdash B \leq B'$, *then* $\Gamma_1, x : C', \Gamma_2 \vdash A \leq A'$ *implies* $\Gamma_1, x : C', \Gamma_2 \vdash B \leq B'$.

Proof. A trivial application of Proposition 4.16 □

Next lemma shows the preservation under bound change of the "unicity of branch" property of overloaded types.

LEMMA 4.18 (Preservation of branch unicity by term bound change).
Let $\Gamma \equiv \Gamma_1, x{:}\, C, \Gamma_2$ *and* $\Gamma' \equiv \Gamma_1, x{:}\, C', \Gamma_2$. *For every set* $\{A_i \mid i \in I\}$ *of closed types,*

$$\forall A\ s.t.\ Fv(A) \subseteq Dom(\Gamma).(\ (\forall i \in I.\ \Gamma \not\vdash A \leq A_i) \\ \vee (\exists! i \in I.\ \Gamma \vdash A \leq A_i\ \wedge\ \forall j \in I\ \Gamma \vdash A \leq A_j\ \Rightarrow\ \Gamma \vdash A_i \leq A_j))$$

implies

$$\forall A\ s.t.\ Fv(A) \subseteq Dom(\Gamma').(\ (\forall i \in I.\ \Gamma' \not\vdash A \leq A_i) \\ \vee (\exists! i \in I.\ \Gamma' \vdash A \leq A_i\ \wedge\ \forall j \in I\ \Gamma' \vdash A \leq A_j\ \Rightarrow\ \Gamma' \vdash A_i \leq A_j))$$

Proof. Use the same technique as Lemma 4.9 and use Proposition 4.16 instead of Lemmas 4.7 and 4.6 □

PROPOSITION 4.19 (Bound narrowing). *Let* $\Gamma_1 \vdash A' \leq A$ *and* $\Gamma_1 \vdash A, A' : \star$. *Then* $\Gamma_1, x : A, \Gamma_2 \vdash J$ *implies* $\Gamma_1, x : A', \Gamma_2 \vdash J$.

Proof. By induction on the derivation of the judgment. For the case T-VAR perform an application of of T-SUB. For the case T-Sub use bound change for subtyping (Proposition 4.16). For the case K-OVER use Lemma 4.17, and 4.18.

The remaining cases are easy. □

PROPOSITION 4.20 (Kinding for subtyping). *If* $\Gamma \vdash A, B : K, \Gamma \vdash A \leq B$, *then for every subtyping judgment* $\Gamma' \vdash C \leq D$ *in the derivation of* $\Gamma \vdash A \leq B$, *there exists a kind* K' *such that* $\Gamma' \vdash C, D : K'$.

Proof. First of all note that without loss of generality we can consider only derivations in which there never are two consecutive applications of K-CONV. Then proceed by induction on the depth of the derivation of $\Gamma \vdash A \leq B$ by performing a case analysis on the last applied rule. The cases for S-ApR, S-OVER, and S-ApT are straightforward. The cases S-ApSR and S-ApSL are direct consequence of Lemma 4.11. A more difficult case is the one for S-π (and S-Λ which is similar): consider $A \equiv \pi x{:}\, C_1.D_1$ and $B \equiv \pi x{:}\, C_2.D_2$. From Proposition 4.14 we obtain that $\Gamma, x{:}\, C_i \vdash D_i : \star$ (i=1,2). From Lemma 4.10, we deduce that $\Gamma \vdash C_i : \star$. Therefore it remains to prove that $\Gamma, x : C_2 \vdash D_1 : \star$. This can be obtained by using Proposition 4.19. The result follows by induction hypothesis. □

4.3.5. $\beta_2\Gamma$ strong normalization

In this subsection, we will introduce the $\beta_2\Gamma$-reduction and prove that it is strongly normalizing. This fact implies that the maximal number of $\beta_2\Gamma$-reduction steps from a type A is always finite. We will use this number in the induction measure for the proofs of several important results, including the proof of transitivity elimination.

We begin by showing the β_2 subject reduction, which is also a property needed in the proofs of transitivity elimination and decidability of subtyping.

PROPOSITION 4.21 (β_2 subject reduction). *If* $\Gamma \vdash J,\ J \twoheadrightarrow_{\beta_2} J'$, *then* $\Gamma \vdash J'$.

Proof. This is the classical proof of subject reduction for simply typed λ-calculus performed by induction on the derivation of $\Gamma \vdash J$. It relies on the generation for kinding (Proposition 4.14). $\Box$

Let Γ be a context, the one step Γ-reduction, denoted by, $\rightarrow_\Gamma$, is defined as the compatible closure of the following reduction:

$$\alpha \ \rightarrow_\Gamma \ \Gamma(\alpha)$$

For the proof of $\beta_2\Gamma$ strong normalization we proceed in two steps. First, we prove the β_2 strong normalization, then we use it to prove the $\beta_2\Gamma$ strong normalization. Intuitively, the first assertion holds because β_2-reduction does not introduce new redexes (existing redex may be duplicated or modified). The second result is obtained by associating every a $\beta_2\Gamma$-reduction sequence to a β_2-reduction sequence that binds it, a technique similar to the one introduced in [Che96].

In more detail, β_2 strong normalization is straightforward since it suffices to observe that β_2 reduction concerns only redexes of the form $(\Lambda x{:}A.B)M$ where M is a term. Reduction of such redexes may duplicate redexes in M, but it will not introduce new redexes.

PROPOSITION 4.22 (β_2 strong normalization). *Let* K, A, M *respectively denote a pre-kind, a pre-type, and a pre-term. Then* K, A, M *are* β_2 *strongly normalizing.*

Proof. Consider M. Define a function S from pre-terms to a multiset of natural numbers.

$$S(M) = \{n \mid n \text{ is the size of a } \beta_2 \text{ redex in } M\}$$

where "the size of a β_2 redex" is the total number of symbols in the redex. If $M \rightarrow_{\beta_2} N$, then a redex R in M will be erased and some subredexes $r \in R$ will be duplicated. Therefore, $S(N)$ is obtained from $S(M)$ by replacing one big number by a finite sequence of strictly smaller numbers. By the well-known multiset order, the reduction will terminate. This holds also if we consider A or K instead of M. $\Box$

The Γ-reduction is obviously strongly normalizing for well-formed Γ (circularities are not allowed). The $\beta_2\Gamma$-reduction is the combination of β_2 and Γ-reduction. Note that the combination of two normalizing reductions may be not normalizing (e.g. consider the union of these two rewriting rules: $a \rightarrow b$ and $b \rightarrow a$). In our specific case, the β_2-reduction may increase the number of Γ-redexes, and on the other hand, a Γ-reduction may increase the number of β_2-redex. We prove the $\beta_2\Gamma$ strong normalization by transforming a $\beta_2\Gamma$-reduction into a β_2-reduction by the function $\mathscr{H}$ which takes an expression and returns its Γ-reduction normal form:

$$\mathscr{H}_\Gamma(U) \ \stackrel{def}{=} \ \text{the } \Gamma\text{-normal form of } U$$

Notice that $\mathscr{H}_\Gamma$ is well-defined on well-formed terms, types and kinds. We will omit the subscript Γ and simply write $\mathscr{H}$ when it is clear from the context.

PROPOSITION 4.23 ($\beta_2\Gamma$ strong normalization). *For* $\beta_2\Gamma$*-reduction, we have the following results:*

1.If $\Gamma \vdash K$*, then K is* $\beta_2\Gamma$ *strongly normalizing;*

2.If $\Gamma \vdash A : K$*, then A is* $\beta_2\Gamma$ *strongly normalizing;*
3.If $\Gamma \vdash M : A$*, then M is* $\beta_2\Gamma$ *strongly normalizing.*

Proof. First of all note that β_2 and Γ reductions commute in a very precise way, namely. If $M \rightarrow_\Gamma M_1$ and $M \rightarrow_{\beta_2} M_2$ then there exists N such that $M_1 \rightarrow_{\beta_2} N$ and $M_2 \twoheadrightarrow_\Gamma N$.

$$\begin{tikzcd}
M \arrow[r, "\Gamma"'] \arrow[d, "\beta_2"'] & M_1 \arrow[d, "\beta_2"] \\
M_2 \arrow[r, twoheadrightarrow, "\Gamma"'] & N
\end{tikzcd}$$

Indeed a Γ-reduction does not affect an existing β_2-redex while a β_2-reduction may duplicate an existing Γ-redex or delete it.

Let $U, U_1, U_2, ...$ be a $\beta_2\Gamma$-reduction sequence starting from U and consider a generic $U_i \rightarrow_R U_{i+1}$. If R is Γ then $\mathscr{F}(U_i) = \mathscr{F}(U_{i+1})$.

If R is β_2 then by observing that the β_2-reductum of a Γ-normal-form is a Γ-normal-form and by composing the commutativity property above we obtain that $\mathscr{F}(U_i) \rightarrow_{\beta_2} \mathscr{F}(U_{i+1})$:

$$\begin{tikzcd}
U_i \arrow[r, "\Gamma"'] \arrow[d, "\beta_2"'] & . \arrow[r, "\Gamma"'] \arrow[d, "\beta_2"] & . \arrow[d, "\beta_2"] & \cdots\cdots & . \arrow[r, "\Gamma"'] \arrow[d, "\beta_2"] & \mathscr{F}(U_i) \arrow[d, "\beta_2"] \\
U_{i+1} \arrow[r, twoheadrightarrow, "\Gamma"'] & . \arrow[r, twoheadrightarrow, "\Gamma"'] & . & \cdots\cdots & . \arrow[r, twoheadrightarrow, "\Gamma"'] & \mathscr{F}(U_{i+1})
\end{tikzcd}$$

So for every i either $\mathscr{F}(U_i) \rightarrow_{\beta_2} \mathscr{F}(U_{i+1})$ or $\mathscr{F}(U_i) \rightarrow^0 \mathscr{F}(U_{i+1})$ (a zero step reduction) holds.

Since Γ is strongly normalizing we cannot have an infinite sequence of zero-step reduction (otherwise these would correspond to an infinite sequence of Γ-reductions on the U_i's).

Thus, $\mathscr{F}(U), \mathscr{F}(U_1), \mathscr{F}(U_2), ...$ is a β_2-reduction sequence starting from $\mathscr{F}(U)$ where $\mathscr{F}(U_i) = \mathscr{F}(U_{i+1})$ for some i. Since β_2-reduction is strongly normalizing and the zero-step reductions are finite, then there exists a number n such that $\mathscr{F}(U_n) = \mathscr{F}(U_{n+1}) = \mathscr{F}(U_{n+2}), ...$ This implies that the reduction sequence $U_n, U_{n+1}, ...$ is a Γ-reduction. But Γ-reduction is normalizing, so the sequence $U, U_1, U_2, ...$ must terminate. $\square$

4.4. Admissible Rules

In this section, we prove that the subtyping relation defined for $\lambda\Pi^{\&}$ is a preorder on well-kinded types. More precisely we prove that the rules (S-CONV) and (S-TRANS) (cf. Section 3.3) are *admissible* (see Footnote 7 for definitions) in our system. Both properties are proved by joint induction on the concerned expressions and on the number of $\beta_2\Gamma$-reduction steps (that are finite, since $\beta_2\Gamma$-reduction is strongly normalizing: Proposition 4.23).

TECHNICAL EXCURSUS. *A notable feature of the subtyping system of $\lambda\Pi^{\&}$ (and $\lambda\Pi_{\leq}$) is that the proof of sub-family judgments is never needed in the deduction of the type of a term. Observe that, given $\Gamma \vdash A, B : \star$, the derivation of $\Gamma \vdash A \leq B$ does not contain any instance of the rule S-Λ. Since the typing system uses subtyping only on types (and not on type families, rule T-SUB), then the rule S-Λ can be harmlessly eliminated from the system (actually, we did not include it in Appendix A.1: the rules in the appendix really define a sub**typing** relation). In other words, while in $\lambda P_{\leq}$ the rule S-Λ is necessary to deduce subtyping on types (when the types are applications of type families), in $\lambda\Pi^{\&}$ (and $\lambda\Pi_{\leq}$) this rule is only used in the deductions of sub-*family *judgments such as $\Gamma \vdash \Lambda x_1{:}\,A_1.\Lambda x_2{:}\,A_2 \ldots \Lambda x_n{:}\,A_n.B \leq \Lambda x_1{:}\,A'_1.\Lambda x_2{:}\,A'_2 \ldots \Lambda x_n{:}\,A'_n.B'$.*

We decided to keep it in our system only because $\lambda P_{\leq}$ has it. With this rule we can show the equivalence of Property 3.1 and state it for all A, B such that $\Gamma \vdash A, B : K$ (instead just for all A, B such that $\Gamma \vdash A, B : \star$).

We ignore this rule in future. Although it is present in several type systems, the rule S-Λ is somewhat anomalous, since it defines true sub-family relations (in which expressions could possibly be kinded by $\Pi x_1{:}\,A_1 ... \Pi x_n{:}\,A_n.\star$ with $n \geq 1$). In this sense (and under the assumption that typing matters more than subtyping), the rules S-ApR, S-ApSL, and S-ApSR are much more reasonable and intuitive since they confine subtyping to pre-types, even in presence of type families.

This feature, the non-utility of the S-Λ, also simplifies the meta-theoretic study. One of its consequences is that we need not *to prove that the general subtyping family application rule S^{AC}-app is admissible in $\lambda\Pi^{\&}$. In the rest of this article we will heavily use reflexivity and transitivity of subtyping (that we prove in this section), but we will not need the subtyping family application property. The only case in which the admissibility of this rule is needed is to prove Property 3.1. That is the reason why we prove the admissibility of this rule for $\lambda\Pi_{\leq}$ (Proposition 6.2) but not for $\lambda\Pi^{\&}$.* $\Box$

Let Γ be a context and U an expression. We denote by *MaxRed*$_\Gamma(U)$ the maximal length of a $\beta_2\Gamma$-reduction starting from U, and by *Size*(U) the number of distinct symbols appearing in U (so for example *Size*$(\lambda x : \alpha.x) = 5$ since we have λ, :, ., x, and α.)

4.4.1. Admissibility of Reflexivity

PROPOSITION 4.24 (Admissibility of Reflexivity). *If $A =_\beta B \wedge \Gamma \vdash A, B : \star$, then $\Gamma \vdash A \leq B$.*

Proof. Since A, B are well kinded types, they can only have one the following forms

$$\begin{array}{ll} \alpha M_1..M_n & n \geq 0 \\ \pi x{:}C.D & \\ (\Lambda x{:}C.D)M_1..M_n & n \geq 1 \\ \{\pi x{:}A_i.B_i\}_{i \in I} & \end{array}$$

Let Γ be a context and A, B two types such that $\Gamma \vdash A, B : \star$. Define the induction measure *Weight*$_\Gamma(A, B)$ as the pair:

$$\textit{Weight}_\Gamma(A,B) =_{def} < \textit{MaxRed}_\Gamma(A) + \textit{MaxRed}_\Gamma(B), \textit{Size}(A) + \textit{Size}(B) >$$

and use the lexicographical order for pairs (most significative component on the left).

By induction on *Weight*$_\Gamma(A, B)$ and examination of all possible cases. $\Box$

With reflexivity, the bound β equivalence property becomes a special case of bound narrowing.

PROPOSITION 4.25 (Bound β-equivalence). *Let* $A =_\beta A'$ *and* $\Gamma_1 \vdash A, A' : K$. *Then:*

$$\begin{array}{rcl} \Gamma_1, x : A, \Gamma_2 \vdash J & \Rightarrow & \Gamma_1, x : A', \Gamma_2 \vdash J \\ \Gamma_1, \alpha \leq A : K, \Gamma_2 \vdash J & \Rightarrow & \Gamma_1, \alpha \leq A' : K, \Gamma_2 \vdash J \end{array}$$

Proof. By induction on the derivation of the judgment. □

4.4.2. Admissibility of Transitivity

In order to prove that the rule

$$\frac{\Gamma \vdash A, B, C : \star \quad \Gamma \vdash A \leq C \quad \Gamma \vdash C \leq B}{\Gamma \vdash A \leq B}\ \textit{trans}$$

is admissible in $\lambda\Pi^{\&}$ we consider the subtyping system extended with the above transitivity rule (we denote it by $\lambda\Pi^{\&}_t$ and judgments provable in the extended system by $\vdash_t$) and we perform a transitivity-elimination process. Namely, we prove that for every derivation in $\lambda\Pi^{\&}_t$ there exists a derivation in $\lambda\Pi^{\&}$ for the same judgment. The method is essentially a process of transforming transitivity applications into derivations in which transitivity occurs only in a smaller degree, as it is usual in cut elimination processes. Therefore, it is necessary to define a well-founded measure over transitivity applications, and show that in each step of transformation, this measure will reduce.

We associate to every application of the transitivity rule

$$\frac{\Gamma \vdash A, B, C : \star \quad \Gamma \vdash A \leq B \quad \Gamma \vdash B \leq C}{\Gamma \vdash A \leq C}$$

the lexicographically ordered measure $\textit{Weight}_\Gamma(A, B, C)$ defined as

$$< \textit{MaxRed}_\Gamma(A) + \textit{MaxRed}_\Gamma(B) + \textit{MaxRed}_\Gamma(C), \textit{Size}(A) + \textit{Size}(B) + \textit{Size}(C) >$$

PROPOSITION 4.26 (Transitivity elimination in $\lambda\Pi^{\&}_t$). *If* $\Gamma \vdash_t A \leq B$, *then* $\Gamma \vdash A \leq B$.

Proof. The proof is done by induction on the number of applications of transitivity appearing in a derivation.

The inductive case is straightforward: if in a given derivation there are $n > 1$ applications of transitivity then consider any subderivation containing exactly one transitivity application. By induction we can transform it into a transitivity free derivation. Thus the global derivation has now $n - 1$ transitivity applications. The result follows by using the induction hypothesis once more.

So let us consider the case in which there exactly is one application of the transitivity rule. Consider the subderivation ending by the transitivity.

$$\frac{\Gamma \vdash A', B', C' : \star \quad \Gamma \vdash A' \leq B' \quad \Gamma \vdash B' \leq C'}{\Gamma \vdash A' \leq C'}\ \textit{trans}$$

The derivations of $\Gamma \vdash A' \leq B'$ and $\Gamma \vdash B' \leq C'$ are transitivity-free, i.e. they are derivations in $\lambda\Pi^{\&}$, whereas $\Gamma \vdash_t A' \leq C'$ is a $\lambda\Pi^{\&}_t$ judgment.

We show by induction on *Weight* that this derivation can be transformed into a transitivity free derivation.

We proceed by case analysis of the last pair of rules used to derive the premises of the transitivity rule.

<u>CASE</u> (S-π, S-π). The derivation must end by

$$\frac{JK \quad \dfrac{\Gamma \vdash A_2 \leq A_1 \quad \Gamma, x:A_2 \vdash B_1 \leq B_2}{\Gamma \vdash \pi x{:}A_1.B_1 \leq \pi x{:}A_2.B_2}\,\text{S-}\pi \quad \dfrac{\Gamma \vdash A_3 \leq A_2 \quad \Gamma, x:A_3 \vdash B_2 \leq B_3}{\Gamma \vdash \pi x{:}A_2.B_2 \leq \pi x{:}A_3.B_3}\,\text{S-}\pi}{\Gamma \vdash_t \pi x{:}A_1.B_1 \leq \pi x{:}A_3.B_3}\,\textit{trans}$$

where JK is the kinding judgment $\Gamma \vdash \pi x{:}A_1.B_1, \pi x{:}A_2.B_2, \pi x{:}A_3.B_3 : \star$.

This derivation can be transformed into

$$\frac{\dfrac{JK1 \quad \Gamma \vdash A_3 \leq A_2 \quad \Gamma \vdash A_2 \leq A_1}{\Gamma \vdash_t A_3 \leq A_1} \quad \dfrac{JK2 \quad \Gamma, x{:}A_3 \vdash B_1 \leq B_2 \quad \Gamma, x{:}A_3 \vdash B_2 \leq B_3}{\Gamma, x{:}A_3 \vdash_t B_1 \leq B_3}}{\Gamma \vdash_t \pi x{:}A_1.B_1 \leq \pi x{:}A_3.B_3}$$

where $JK1 \equiv \Gamma \vdash A_1, A_2, A_3 : \star$ and $JK2 \equiv \Gamma, x : A_3 \vdash B_1, B_2, B_3 : \star$.

The derivability of the judgments $JK1, JK2$ and $\Gamma, x : A_3 \vdash B_1 \leq B_2$ can be obtained as follows.

$\Gamma \vdash \pi x{:}A_1.B_1, \pi x{:}A_2.B_2, \pi x{:}A_3.B_3 : \star$
$\quad \Rightarrow \Gamma, x : A_1 \vdash B_1 : \star \wedge \Gamma, x : A_2 \vdash B_2 : \star \wedge \Gamma, x : A_3 \vdash B_3 : \star$ Prop. 4.14
$\quad \Rightarrow \Gamma \vdash A_1, A_2, A_3 : \star$ Prop. 4.10

$\Gamma \vdash A_3, A_2 : \star \wedge \Gamma \vdash A_3 \leq A_2 \wedge \Gamma, x : A_2 \vdash B_1 \leq B_2$
$\quad \Rightarrow \Gamma, x : A_3 \vdash B_1 \leq B_2$ Prop. 4.16

$\Gamma \vdash A_3, A_2 : \star \wedge \Gamma \vdash A_3 \leq A_2 \wedge \Gamma, x : A_2 \vdash B_2 : \star$
$\quad \Rightarrow \Gamma, x : A_3 \vdash B_2 : \star$ Prop. 4.19

$\Gamma \vdash A_3, A_2 : \star \wedge \Gamma \vdash A_3 \leq A_1 \wedge \Gamma, x : A_1 \vdash B_1 : \star$
$\quad \Rightarrow \Gamma, x : A_3 \vdash B_1 : \star$ Prop. 4.19

In the last implication $\Gamma \vdash A_3 \leq A_1$ follows by induction hypothesis from $\Gamma \vdash A_1, A_2, A_3 : \star$, $\Gamma \vdash A_3 \leq A_2$, and $\Gamma \vdash A_2 \leq A_1$.

In conclusion, $\Gamma \vdash A_1, A_2, A_3 : \star, \Gamma, x : A_3 \vdash B_1, B_2, B_3 : \star$ and $\Gamma, x : A_3 \vdash B_1 \leq B_2$ are all derivable without transitivity. Furthermore, we have two new subderivations in which the transitivity appears only once at the end, the sizes of whose types are strictly smaller than those of the original transitivity application and where maximal $\beta_2\Gamma$-reduction steps do not increase (note indeed that $\textit{MaxRed}_{\Gamma,x:F}(E) = \textit{MaxRed}_{\Gamma}(E)$ for every Γ, F, and E) . So $\textit{Weight}_{\Gamma}(A_1, A_2, A_3)$ and $\textit{Weight}_{\Gamma,x_3:A}(B_1, B_2, B_3)$ are strictly less than $\textit{Weight}_{\Gamma}(\pi x{:}A_1.B_1, \pi x{:}A_2.B_2, \pi x{:}A_3.B_3)$. Finally, by induction hypothesis the transitivity application in the new derivations can be eliminated.

<u>CASE</u> (S-ApR, S-ApR). By transitivity of β-conversion.

<u>CASE</u> (S-ApSL, _) The derivation must end by

$$\frac{JK \quad \dfrac{\Gamma \vdash (B[x := M_1])M_2..M_n \leq C}{\Gamma \vdash (\Lambda x{:}A.B)M_1..M_n \leq C}\,\text{S-ApSL} \quad \Gamma \vdash C \leq D}{\Gamma \vdash_t (\Lambda x{:}A.B)M_1..M_n \leq D}\,\textit{trans}$$

where $JK \equiv \Gamma \vdash (\Lambda x{:}A.B)M_1..M_n, C, D : \star$. By β_2 subject-reduction (Lemma 4.21), $(B[x := M_1])M_2..M_n$ is well-kinded in the context Γ. So the above derivation can be transformed into

$$\cfrac{\cfrac{JK' \quad \Gamma \vdash (B[x := M_1])M_2..M_n \leq C \quad \Gamma \vdash C \leq D}{\Gamma \vdash_t (B[x := M_1])M_2..M_n \leq D}\ \textit{trans}}{\Gamma \vdash_t (\Lambda x{:}A.B)M_1..M_n \leq D}\ \text{S-ApSL}$$

where $JK' \equiv \Gamma \vdash (B[x := M_1])M_2..M_n, D, C : \star$. The sizes of the types in the transitivity application may increase, but the maximal number of steps of $\beta_2\Gamma$-reduction decreases since $(\Lambda x{:}A.B)M_1..M_n \to_{\beta_2} (B[x := M_1])M_2..M_n$. So the derivation measure *Weight* decreases for the new transitivity application. The result follows by induction hypothesis.

<u>CASE</u> (_, S-ApSR). Similar.

<u>CASE</u> (S-ApT, _). Similar. Just note that *Weight* decreases because there is a Γ-reduction:

$$\alpha M_1..M_n \ \to_\Gamma \ \Gamma(\alpha)M_1..M_n$$

<u>CASE</u> (S-ApR, S-ApT). The derivation must end by

$$\cfrac{JK \quad \cfrac{M_1 =_\beta M_1' \quad \cdots \quad M_n =_\beta M_n'}{\Gamma \vdash \alpha M_1..M_n \leq \alpha M_1'..M_n'}\ \text{S-ApR} \quad \cfrac{\Gamma \vdash \Gamma(\alpha)M_1'..M_n' \leq C}{\Gamma \vdash \alpha M_1'..M_n' \leq C}\ \text{S-ApT}}{\Gamma \vdash_t \alpha M_1..M_n \leq C}\ \textit{trans}$$

where $JK \equiv \Gamma \vdash \alpha M_1..M_n, \alpha M_1'..M_n', C : \star$. From the kinding assumption JK and from the observation that α and $\Gamma(\alpha)$ have the same kind it follows that

$$\Gamma \vdash \Gamma(\alpha)M_1..M_n, \Gamma(\alpha)M_1'..M_n', C : \star$$

By the reflexivity of subtyping (Proposition 4.24), the judgment $\Gamma \vdash \Gamma(\alpha)M_1..M_n \leq \Gamma(\alpha)M_1'..M_n'$ is derivable. Therefore, we have a derivation ending by

$$\cfrac{\cfrac{JK' \quad \Gamma \vdash \Gamma(\alpha)M_1..M_n \leq \Gamma(\alpha)M_1'..M_n' \quad \Gamma \vdash \Gamma(\alpha)M_1'..M_n' \leq C}{\Gamma \vdash_t \Gamma(\alpha)M_1..M_n \leq C}\ \textit{trans}}{\Gamma \vdash_t \alpha M_1..M_n \leq C}\ \text{S-ApT}$$

where $JK' \equiv \Gamma \vdash \Gamma(\alpha)M_1..M_n, \Gamma(\alpha)M_1'..M_n', C : \star$. Again the *Weight* measure decreases because of the Γ-reduction.

<u>CASE</u> (S-ApSR, S-ApSL). The derivation must end by

$$\cfrac{JK \quad \cfrac{\Gamma \vdash C \leq B[x := M_1]M_2..M_n}{\Gamma \vdash C \leq (\Lambda x{:}A.B)M_1..M_n}\ \text{S-ApSR} \quad \cfrac{\Gamma \vdash B[x := M_1]M_2..M_n \leq C'}{\Gamma \vdash (\Lambda x{:}A.B)M_1..M_n \leq C'}\ \text{S-ApSL}}{\Gamma \vdash_t C \leq C'}\ \textit{trans}$$

where $JK \equiv \Gamma \vdash C, (\Lambda x{:}A.B)M_1..M_n, B[x := M_1]M_2..M_n : \star$. The derivation can be transformed into

$$\cfrac{JK' \quad \Gamma \vdash C \leq B[x := M_1]M_2..M_n \quad \Gamma \vdash B[x := M_1]M_2..M_n \leq C'}{\Gamma \vdash_t C \leq C'}\ \textit{trans}$$

where $JK' \equiv \Gamma \vdash C, B[x := M_1]M_2..M_n, C' : \star$. The *Weight* decreases because of the β_2-reduction.

CASE (S-OVER,S-OVER). Suppose that the last step of the derivation is

$$\frac{JK \quad \Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\in I} \leq \{\pi x{:}E_h.F_h\}_{h\in H} \quad \Gamma \vdash \{\pi x{:}E_h.F_h\}_{h\in H} \leq \{\pi x{:}C_j.D_j\}_{j\in J}}{\Gamma \vdash_t \{\pi x{:}A_i.B_i\}_{i\in I} \leq \{\pi x{:}C_j.D_j\}_{j\in J}}\ \textit{trans}$$

where $JK \equiv \Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\in I}, \{\pi x{:}E_h.F_h\}_{h\in H}, \{\pi x{:}C_j.D_j\}_{j\in J} : \star$.

Since this is the only application of the transitivity rule, the two assumptions can only be derived by the S-OVER rule. That is,

$$\frac{\forall j \in J \ \ \Gamma \vdash \pi x{:}E_{\phi(j)}.F_{\phi(j)} \leq \pi x{:}C_j.D_j}{\Gamma \vdash \{\pi x{:}E_h.F_h\}_{h\in H} \leq \{\pi x{:}C_j.D_j\}_{j\in J}}\ \text{S-OVER}$$

and

$$\frac{\forall h \in H \ \ \Gamma \vdash \pi x{:}A_{\psi(h)}.B_{\psi(h)} \leq \pi x{:}E_h.F_h}{\Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\in I} \leq \{\pi x{:}E_h.F_h\}_{h\in H}}\ \text{S-OVER}$$

with both $\phi : J \to H$ and $\psi : H \to I$ total.

For every $j \in J$,

$$\Gamma \vdash \pi x{:}A_{\psi(\phi(j))}.B_{\psi(\phi(j))} \leq \pi x{:}E_{\phi(j)}.F_{\phi(j)} \ \wedge \ \Gamma \vdash \pi x{:}E_{\phi(j)}.F_{\phi(j)} \leq \pi x{:}C_j.D_j$$

Since *Weight* has decreased we can apply the induction hypothesis obtaining that for every $j \in J$

$$\Gamma \vdash \pi x{:}A_{\psi(\phi(j))}.B_{\psi(\phi(j))} \leq \pi x{:}C_j.D_j$$

which means that

$$\forall j \in J \ \exists i \in I \ \ \Gamma \vdash \pi x{:}A_i.B_i \leq \pi x{:}C_j.D_j$$

The result follows by S-OVER.

□

COROLLARY 4.27 (Admissibility of transitivity). *If* $\Gamma \vdash A, B, C : \star$, $\Gamma \vdash A \leq B$, *and* $\Gamma \vdash B \leq C$, *then* $\Gamma \vdash A \leq C$.

4.5. Subject Reduction

In this section, we show the generation for typing and prove subject reduction.

This result relies on the admissibility of transitivity stated in the previous section. Indeed, the first step in proving subject reduction consists in proving that every typing derivation can be transformed into a derivation where there are no consecutive applications of the subsumption rule. This follows straightforwardly from the transitivity of subtyping since whenever there are two consecutive applications of subsumption such as

$$\frac{\dfrac{\Gamma \vdash M : A \quad \Gamma \vdash A \leq B \quad \Gamma \vdash A, B : \star}{\Gamma \vdash M : B}\ \text{T-SUB} \quad \Gamma \vdash B \leq C \quad \Gamma \vdash B, C : \star}{\Gamma \vdash M : C}\ \text{T-SUB}$$

then it is possible to deduce from the transitivity of subtyping that $\Gamma \vdash A \leq C$ and, therefore, replace them by

$$\frac{\Gamma \vdash M : A \quad \Gamma \vdash A \leq C \quad \Gamma \vdash A, C : \star}{\Gamma \vdash M : C} \text{ T-SUB}$$

Generation for typing describes the information we can infer about a type from a provable typing judgment.

PROPOSITION 4.28 (Generation for typing).

$\Gamma \vdash x : C \Rightarrow \Gamma \vdash \Gamma(x) \leq C$

$\Gamma \vdash \lambda x{:}A.M : C \Rightarrow \exists B \; s.t. \; \Gamma, x : A \vdash M : B \;\wedge\; \Gamma \vdash \pi x{:}A.B \leq C$

$\Gamma \vdash MN : C \Rightarrow \exists A, B \; s.t. \; \Gamma \vdash M : \pi x{:}A.B \;\wedge\; \Gamma \vdash N : A \;\wedge\; \Gamma \vdash B[x := N] \leq C$

$\Gamma \vdash M \bullet N : C \Rightarrow \Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i \leq n} \;\wedge\; (\exists i \leq n. \, \Gamma \vdash N : A_i \wedge \Gamma \vdash B_i[x := N] \leq C)$

$\Gamma \vdash M_1 \&^A M_2 : C \Rightarrow \Gamma \vdash M_1 : \{\pi x{:}\, A_i.B_i\}_{1=1..n-1} \wedge \Gamma \vdash M_2 : \pi x{:}A_n.B_n \;\wedge\; \Gamma \vdash A \leq C$
where $A = \{\pi x{:}\, A_i.B_i\}_{i=1..n}$

Subject reduction is one of the main concerns in the study of dependent types with subtyping. Since subtyping is separated from other judgments the proof is quite simple.

PROPOSITION 4.29 (Subject reduction). *If* $\Gamma \vdash J$ *and* $J \twoheadrightarrow_\beta J'$, *then* $\Gamma \vdash J'$.

Proof. It is enough to prove the one step case, since the result follows by induction on the number of the steps. We prove the one step case by induction on the derivation of the judgment $\Gamma \vdash J$ and performing a case analysis on the reduction. Without loss of generality, we assume that there are not two consecutive applications of subsumption.

We show the most significative cases. The others are either similar or straightforward.

<u>CASE</u> $J \equiv \Gamma \vdash (\lambda x{:}A.M)N : B \;\wedge\; (\lambda x{:}A.M)N \to_{\beta_1} M[x := N]$.
By Proposition 4.28 there exist C and D such that $\Gamma \vdash \lambda x{:}\, A.M : \pi x{:}\, C.D, \Gamma \vdash N : C$, and

$$\Gamma \vdash D[x := N] \leq B \tag{7}$$

We apply once more Proposition 4.28 and we obtain that there exists F such that $\Gamma, x{:}\, A \vdash M : F$ and $\Gamma \vdash \pi x{:}\, A.F \leq \pi x{:}\, C.D$. From this last judgment we deduce that $\Gamma \vdash C \leq A$ and that $\Gamma, x : C \vdash F \leq D$. By Lemma 4.16 we obtain

$$\Gamma, x : A \vdash F \leq D \tag{8}$$

From $\Gamma, x{:}\, A \vdash M : F$ we deduce $\Gamma, x{:}\, A \vdash \star$ (Proposition 4.10) and thus $\Gamma \vdash A : \star$ (rule F-TERM). Since $\Gamma \vdash N : C$ then $\Gamma \vdash C : \star$ (Proposition 4.13). We can apply T-SUB to obtain $\Gamma \vdash N : A$ and by a weakening (third point of Proposition 4.10).

$$\Gamma, x{:}\, A \vdash N : A \tag{9}$$

By using Proposition 4.13 and generation for kinding (Proposition 4.14) we obtain that $\Gamma, x{:}\, A \vdash F, D : \star$. We can then apply T-SUB to (8) and $\Gamma, x{:}\, A \vdash M : F$ and deduce

$$\Gamma, x{:}\, A \vdash M : D \tag{10}$$

Finally by (9), (10), and $\Gamma, x{:}\, A \vdash x : A$ we can apply the Substitution Lemma 4.11 and obtain

$$(\Gamma, x{:}\, A)[x := N] \vdash M[x := N] : D[x := N]$$

But $\Gamma, x{:}\, A \vdash \star$ therefore by Definition 4.5 $(\Gamma, x{:}\, A)[x := N] = \Gamma$. By Proposition 4.13 $\Gamma \vdash D[x := N] : \star$. Therefore by (7) we can apply T-SUB and deduce $\Gamma \vdash M[x := N] : B$, that is the result.

<u>CASE</u> $\Gamma \vdash \Pi x{:}A.K$ and $A \rightarrow_\beta A'$.
In this case, we have a derivation ending by

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

It follows from the context property (Proposition 4.10) that $\Gamma \vdash A : \star$ is derivable by a proof less deep than the derivation for $\Gamma, x : A \vdash K$. Therefore, we can apply the induction hypothesis and get $\Gamma \vdash A' : \star$. By bound β-equivalence (Proposition 4.25) and the fact that $\Gamma, x : A \vdash K$ and $A =_\beta A'$, we get $\Gamma, x : A' \vdash K$, the result $\Gamma \vdash \Pi x{:}A'.K$ follows.

<u>CASE</u> $\Gamma \vdash (M_1 \&^{\{\pi x : A_i . B_i\}_{i=1..n}} M_2) \bullet N : C$ and $(M_1 \&^{\{\pi x : A_i . B_i\}_{i=1..n}} M_2) \bullet N \rightarrow_{\beta\&} M_1 \bullet N$.
Let $A \equiv \{\pi x{:}\, A_i.B_i\}_{i=1..n}$. From $\Gamma \vdash (M_1 \&^A M_2) \bullet N : C$ we can deduce that $\Gamma \vdash (M_1 \&^A M_2) : A$, $\Gamma \vdash M_1 : \{\pi x{:}\, A_i.B_i\}_{i=1..n-1}$, and

$$\Gamma \vdash A : \star \tag{11}$$

Acting as in the first case of this proof we can apply Proposition 4.28 twice and couple it with the subsumption rule to deduce that there exists $h \in [1..n]$ such that $\Gamma \vdash N : A_h$ and $\Gamma \vdash B_h[x := N] \leq C$.

From (11) we deduce that

$$\Gamma, x{:}\, A_h \vdash B_h : \star \tag{12}$$

Therefore we have that $\Gamma, x{:}\, A_h \vdash \star$ (Proposition 4.10) and thus $x \notin Dom(\Gamma)$ (rule F-TERM).

Since $\Gamma \vdash N : A_h$ and $\Gamma, x{:}\, A_h \vdash \star$ then by a weakening (third point of Proposition 4.10) we deduce $\Gamma, x{:}\, A_h \vdash N : A_h$. We can thus apply the Substitution Lemma 4.11 to (12) and obtain

$$\Gamma \vdash B_h[x := N] : \star \tag{13}$$

By definition of $\beta^\&$ reduction we have that there exists $j \in [1..n-1]$ such that $\Gamma \vdash N : A_j$ and

$$\forall A_i \; \Gamma \vdash N : A_i \Rightarrow A_j \leq A_i \tag{14}$$

Therefore

$$\begin{array}{rll}
\Gamma \vdash N : A_h & \Rightarrow \; \Gamma \vdash A_j \leq A_h & \text{By (14)} \\
& \Rightarrow \; \Gamma, x : A_j \vdash B_j \leq B_h & \text{By covariance to deduce (11)} \\
& \Rightarrow \; \Gamma[x := N] \vdash B_j[x := N] \leq B_h[x := N] & \text{Lemma 4.6} \\
& \Rightarrow \; \Gamma \vdash B_j[x := N] \leq B_h[x := N] & \text{Since } x \notin Dom(\Gamma)
\end{array}$$

Finally we have

$$\begin{array}{lll} M_1 : \{\pi x{:}\, A_i.B_i\}_{i=1..n-1} \ \wedge\ \Gamma \vdash N : A_j \Rightarrow & & \\ \qquad \Rightarrow M_1 \bullet N : B_j[x := N] & & \text{T-OAPP} \\ \qquad \Rightarrow M_1 \bullet N : B_h[x := N] & & \text{By (13) and T-SUB} \\ \qquad \Rightarrow M_1 \bullet N : C & & \text{By T-SUB and Proposition 4.13} \end{array}$$

□

4.6. Conservativity

In this section, we show that $\lambda\Pi^{\&}$ is a conservative extension both of $\lambda\Pi_{\leq}$ and of $\lambda\&$. In other words, let $\Gamma \vdash J$ be a judgment derivable in $\lambda\Pi^{\&}$: if expressions in J are free of overloaded types and terms, then $\Gamma \vdash J$ is derivable in $\lambda\Pi_{\leq}$; if expressions in J are free of dependent types and terms, then $\Gamma \vdash J$ is derivable in $\lambda\&$.

This is not a straightforward property since for example a derivation ending with a judgment free of overloading terms and types may contain overloaded terms or types in some judgments in the middle of the derivation. So a judgment not provable in $\lambda\Pi_{\leq}$ might be provable in $\lambda\Pi^{\&}$ even if the judgment is free of overloading. For example we might have two overloading free types A and B, and a type C containing overloaded types such that $A \leq C \leq B$ is provable in $\lambda\Pi^{\&}$ but $A \leq B$ is not provable in $\lambda\Pi_{\leq}$ (this happens for example when $F_{\leq}$ is extended by recursive types: see [Ghe93]).

4.6.1. Conservativity with respect to $\lambda\Pi_{\leq}$

DEFINITION 4.30 (Free of overloading). *We say a pre-expression U is* free of overloading *if it does not contain overloaded terms (overloaded function and application) or types.*

We use $=^S_\beta$ to denote β-conversion in the system S, and $\vdash^S$ to denote judgments provable in system S.

LEMMA 4.31 (Conservativity of conversion). *Given the pre-expressions U, V, if U, V are free of overloading, then*

1. $U \to_\beta U' \quad \Rightarrow\ $ *U' is free of overloading*
2. $U =^{\lambda\Pi^{\&}}_\beta V \ \Rightarrow\ U =^{\lambda\Pi_{\leq}}_\beta V$

PROPOSITION 4.32 (Conservativity of judgment). *Suppose that Γ is a context in which all expressions are free of overloading, J is either a typing , or kinding, or context formation, or subtyping judgment. Then,*

$$\Gamma \vdash^{\lambda\Pi^{\&}} J \ \Rightarrow\ \Gamma \vdash^{\lambda\Pi_{\leq}} J$$

Proof. The proof is very easy. This is due to the fact that the rules that define our system satisfy the subformula property. This is important in particular for the subtyping rules, which are transitivity free.

Therefore we can first prove the assertion on the subtyping system by induction on the depth of the derivation. The interesting cases are S-ApR and S-Λ where there are β conversions and that are straightforwardly proved by using Lemma 4.31 (2).

And finally the result is proved by a simultaneous induction on the depth of the derivations of context formation, typing and kinding judgments. □

4.6.2. Conservativity with respect to $\lambda\&$

In this subsection we prove that $\lambda\Pi^{\&}$ is a conservative extension of $\lambda\&$-calculus, whose pre-expressions are defined as follows

$$
\begin{array}{lcl}
M & ::= & x \mid \lambda x{:}A.M \mid MM \mid \varepsilon \mid \ M\&^{A}M \mid M \bullet M \\
A & ::= & \alpha \mid A \to A \mid \{A \to A, ..., A \to A\} \\
K & ::= & \star \\
\Gamma & ::= & <> \ \mid \Gamma, x : A \mid \Gamma, \alpha \leq A : K
\end{array}
$$

In the rest of this section, $A \to B$ denotes the type $\pi x{:}\, A.B$ where x is not free in B.

DEFINITION 4.1. Let Γ be a $\lambda\Pi^{\&}$ context, we use $SD(\Gamma)$ and $TD(\Gamma)$ to denote respectively the set of all subtyping declarations, and the set of all typing declarations in Γ. More precisely we have:

$$
SD(\Gamma) = \begin{cases} <> & \text{for } \Gamma \equiv <> \\ SD(\Gamma'), \alpha \leq A{:}\star & \text{for } \Gamma \equiv \Gamma', \alpha \leq A : \star \\ SD(\Gamma') & \text{for } \Gamma \equiv \Gamma', J \text{ and } J \not\equiv \alpha \leq A : \star \end{cases}
$$

$$
TD(\Gamma) = \begin{cases} <> & \text{for } \Gamma \equiv <> \\ TD(\Gamma'), x{:}\, A & \text{for } \Gamma \equiv \Gamma', x{:}\, A \\ TD(\Gamma') & \text{for } \Gamma \equiv \Gamma', J \text{ and } J \not\equiv x{:}\, A \end{cases}
$$

In order to compare $\lambda\&$ with $\lambda\Pi^{\&}$ we give a definition of $\lambda\&$, which results in a system that is slightly different from the one of [CGL95]. There are two differences between "standard" $\lambda\&$ and $\lambda\Pi^{\&}$ and they both concern the subtyping of atomic types. The first is that, in $\lambda\&$, it is possible to have in a context two subtyping declarations for the same lower bound (such as $\alpha \leq A, \alpha \leq B$) while this is not allowed in $\lambda\Pi^{\&}$. The second difference is that, in $\lambda\&$, the upper bound, say A, occurring in a subtyping declaration $\alpha \leq A$, must be an atomic type, while in $\lambda\Pi^{\&}$, A can be any type.[24] The variant of $\lambda\&$ presented below takes the $\lambda\Pi^{\&}$ approach and allows at most one subtyping declaration $\alpha \leq A$ for each atomic type α but allows A to be any type. Finally, since $\lambda\&$ does not have dependent types, then no type contains terms. Therefore, contexts in subtyping judgment do not need to contain typing declarations. Thus, let $\mathscr{S}$ be a generic context of subtyping declarations of the form $\alpha \leq A{:}\star$ (where A is a $\lambda\&$ pre-type). The subtyping relation induced for $\lambda\&$-calculus by $\mathscr{S}$ is defined as follows:

Subtyping

$$
\text{S}^{\lambda\&}\text{-REFL} \quad \frac{}{\mathscr{S} \vdash^{\lambda\&} \alpha \leq \alpha} \qquad \text{S}^{\lambda\&}\text{-}\!\to \quad \frac{\mathscr{S} \vdash^{\lambda\&} A' \leq A \qquad \mathscr{S} \vdash^{\lambda\&} B \leq B'}{\mathscr{S} \vdash^{\lambda\&} A \to B \leq A' \to B'}
$$

$$
\text{S}^{\lambda\&}\text{-TRANS} \quad \frac{\mathscr{S} \vdash^{\lambda\&} \Gamma(\alpha) \leq A}{\mathscr{S} \vdash^{\lambda\&} \alpha \leq A} \qquad \text{S}^{\lambda\&}\text{-OVER} \quad \frac{\forall j \in J\ \exists i \in I\ \mathscr{S} \vdash^{\lambda\&} A_i \to B_i \leq A'_j \to B'_j}{\mathscr{S} \vdash^{\lambda\&} \{A_i \to B_i\}_{i\in I} \leq \{A'_j \to B'_j\}_{j\in J}}
$$

The remaining rules of $\lambda\&$-calculus are given below. Note that the T-SUB rule is only connection between subtyping and the rest of the system. Note also that as contexts do not

[24] These two differences are closely related. They both serve to avoid to relate types with different structures (e.g. an overloaded type and a π-type). In this work we followed the theoretically oriented approach of [AC96b] in which there is at most one subtyping declaration for each atomic type. In [CGL95] instead atomic types may form a lattice, since it is a more practically-oriented solution (it allows the so-called "multiple-inheritance"). We believe that an implementation of $\lambda\Pi^{\&}$ should rather use this second solution.

contain kinding declarations, all the atomic types are considered well-formed[25]

Context formation

$$\text{F}^{\lambda\&}\text{-EMPTY}\quad \frac{}{<>\vdash^{\lambda\&}\star} \qquad\qquad \text{F}^{\lambda\&}\text{-TERM}\quad \frac{\Gamma\vdash^{\lambda\&} A:\star \quad x\notin Dom(\Gamma)}{\Gamma, x:A\vdash^{\lambda\&}\star}$$

$$\text{F}^{\lambda\&}\text{-SUBTYPE}\quad \frac{\Gamma\vdash^{\lambda\&} A:\star \quad \alpha\notin Dom(\Gamma)}{\Gamma, \alpha\leq A{:}\star\vdash^{\lambda\&}\star}$$

Kinding

$$\text{K}^{\lambda\&}\text{-VAR}\quad \frac{\Gamma\vdash\star}{\Gamma\vdash^{\lambda\&}\alpha:\star} \qquad\qquad \text{K}^{\lambda\&}\text{-}\!\to\quad \frac{\Gamma\vdash^{\lambda\&} A:\star \qquad \Gamma\vdash^{\lambda\&} B:\star}{\Gamma\vdash^{\lambda\&} A\to B:\star}$$

$$\text{K}^{\lambda\&}\text{-OVER}\quad \frac{\begin{array}{c}\Gamma\vdash\star\\ \forall i\in I.\,\Gamma\vdash^{\lambda\&} A_i\to B_i:\star\\ \forall i,j\in I.\,\Gamma\vdash^{\lambda\&} A_i\leq A_j \;\Rightarrow\; \Gamma\vdash^{\lambda\&} B_i\leq B_j\\ \forall A.\,(\forall i\in I.\,\Gamma\vdash^{\lambda\&} A\not\leq A_i)\;\vee\\ \vee\,(\exists! i\in I.\,\Gamma\vdash^{\lambda\&} A\leq A_i\;\wedge\;\forall j\in I.\;\Gamma\vdash^{\lambda\&} A\leq A_j\Rightarrow\Gamma\vdash^{\lambda\&} A_j\leq A_i)\end{array}}{\Gamma\vdash^{\lambda\&}\{A_i\to B_i\}_{i\in I}:\star}$$

Typing

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

$$\text{T}^{\lambda\&}\text{-}\lambda\quad \frac{\Gamma, x:A\vdash^{\lambda\&} M:B}{\Gamma\vdash^{\lambda\&}\lambda x{:}A.M:A\to B} \qquad \text{T}^{\lambda\&}\text{-\&}\quad \frac{\begin{array}{c}\Gamma\vdash^{\lambda\&}\{A_i\to B_i\}_{i\leq n+1}:\star\\ \Gamma\vdash^{\lambda\&} M:\{A_i\to B_i\}_{i\leq n}\\ \Gamma\vdash^{\lambda\&} N:A_{n+1}\to B_{n+1}\end{array}}{\Gamma\vdash^{\lambda\&} M\&^{\{A_i\to B_i\}_{i\leq n+1}}N:\{A_i\to B_i\}_{i\leq n+1}}$$

$$\text{T}^{\lambda\&}\text{-APP}\quad \frac{\Gamma\vdash^{\lambda\&} M:A\to B\quad \Gamma\vdash^{\lambda\&} N:A}{\Gamma\vdash^{\lambda\&} MN:B}$$

$$\text{T}^{\lambda\&}\text{-OAPP}\quad \frac{\Gamma\vdash^{\lambda\&} M:\{A_i\to B_i\}_{i\leq n}\quad \Gamma\vdash^{\lambda\&} N:A_i}{\Gamma\vdash^{\lambda\&} M\bullet N:B_i}$$

$$\text{T}^{\lambda\&}\text{-SUB}\quad \frac{\Gamma\vdash^{\lambda\&} M:A\quad SD(\Gamma)\vdash^{\lambda\&} A\leq B\quad \Gamma\vdash^{\lambda\&} A,B:\star}{\Gamma\vdash^{\lambda\&} M:B}$$

The notions of reduction are $\beta^{\&}$ and β_1 as defined for $\lambda\Pi^{\&}$. It is easy to verify by using Theorems 4.2.2 and 4.2.4 of [Cas97] that the above definition is equivalent to the one in [CGL95] (modulo the differences on the subtyping of atomic types).

DEFINITION 4.33 (Free of dependent types). *We say that a* $\lambda\Pi^{\&}$ *judgment* $\Gamma\vdash J$ *is* free of dependent types *if lenght*(Γ) = *length*$(SD(\Gamma))$ + *length*$(TD(\Gamma))$ *(that is,* Γ *does not contain kinding declarations of the form* $\alpha{:}K$ *or* $\alpha\leq A:\Pi x{:}A'.K$*) and*

[25] We preferred to maintain the notation "$\alpha\leq A{:}\star$" for subtyping declarations, even though the "$:\star$" could be clearly omitted.

1.if $J \equiv A \leq B$, then A, B are $\lambda\&$ pre-types;
2.if $J \equiv A : \star$, then A is a $\lambda\&$ pre-type;
3.if $J \equiv M : A$, then M is a $\lambda\&$ pre-term and A a $\lambda\&$ pre-type;
4.for all $\alpha \in Dom(\Gamma)$, $\Gamma(\alpha)$ is a $\lambda\&$ pre-type.

PROPOSITION 4.34 (Conservativity of subtyping w.r.t. $\lambda\&$). *If $\Gamma \vdash A \leq B$ is free of dependent types, then $\Gamma \vdash^{\lambda\Pi^{\&}} A \leq B$ implies $SD(\Gamma) \vdash^{\lambda\&} A \leq B$.*

Proof. By induction on the derivation of $\Gamma \vdash^{\lambda\Pi^{\&}} A \leq B$, and performing a case analysis on the last rule of the derivation. □

Similarly, we have

PROPOSITION 4.35 (Conservativity w.r.t. $\lambda\&$). *If $\Gamma \vdash J$ is free of dependent types, then $\Gamma \vdash^{\lambda\Pi^{\&}} J$ implies $\Gamma \vdash^{\lambda\&} J$.*

Proof. Straightforward induction on the derivation of $\Gamma \vdash J$. The case of subsumption requires the use of Proposition 4.34. For the case K-π, note that $\Gamma, x : A \vdash^{\lambda\Pi^{\&}} B : \star$ implies that $\Gamma \vdash^{\lambda\Pi^{\&}} B : \star$. Indeed, it is easy to see that well-formation under a context Γ depends only on the subtyping declarations of Γ, that is, that for every judgment J of the form $\star$ or $A{:}\star$, we have that $\Gamma \vdash^{\lambda\&} J$ implies $SD(\Gamma) \vdash^{\lambda\&} J$. This last observation is used also for the case K-OVER. □

5. STRONG NORMALIZATION

System $\lambda\Pi^{\&}$ is not strong normalizing since it is a conservative extension of $\lambda\&$-calculus that is not strongly normalizing [CGL95]. But, as for $\lambda\&$, strong normalization holds for a subsystem of $\lambda\Pi^{\&}$, that we call $\lambda\Pi^{-}_{\&}$. The study of strong normalization is not undertaken for its own sake, but because in this framework strong normalization implies decidability of subtyping, which in its turn implies the decidability of the type system. Thus thanks to the strong normalization result of this section and the result of Section 7 we are able to show that $\lambda\Pi^{-}_{\&}$ is a (type) decidable subsystem of $\lambda\Pi^{\&}$ (whose type system is not decidable). In this section we adapt to $\lambda\Pi^{-}_{\&}$ the technique developed in [CGL95] to prove strong normalization for $\lambda\&^{-}$ a subsystem of $\lambda\&$.

In order to prove strong normalization in the subsystem $\lambda\&^{-}$ of $\lambda\&$-calculus, [CGL95] introduces a variant of the Tait proof technique [Tai67] (improved by Girard in [Gir87]). Recall that the Tait method consists of the following steps:

1. define a set of terms called reducible set R,
2. show that $M \in R \Rightarrow M \in \mathscr{SN}$, where $\mathscr{SN}$ is the set of strongly normalizing terms,
3. show that $M : A \Rightarrow M \in R$, that is, well-typed terms belong to the reducible set.

The reducible set is a union of sets indexed over types: $R = \cup_{A \in \mathbf{Types}} R_A$. For example for the simply typed λ-calculus R_A is defined inductively as

$$\begin{array}{llll} 1. & M \in R_A & \Leftrightarrow \; M \in \mathscr{SN} & A \text{ is an atomic type} \\ 2. & M \in R_{A \to B} & \Leftrightarrow \; \forall N \in R_A.\, MN \in R_B & \end{array}$$

A naive (and wrong) generalization of the above definition to $\lambda\&$-calculus might be,

$$\begin{array}{llll} 1. & M \in R_A & \Leftrightarrow \; M \in \mathscr{SN} & A \text{ is an atomic type} \\ 2. & M \in R_{A \to B} & \Leftrightarrow \; \forall A' \leq A. \forall N \in R_{A'}.\, MN \in R_B & \\ 3. & M \in R_{\{A_i \to B_i\}_{i \in I}} & \Leftrightarrow \; \forall A' \in [A_i]_{i \in I}. \forall N \in R_{A'}.\, M \bullet N \in R_{B_i} & \end{array}$$

where $[A_i]_{i\in I}$ denotes the set of types that are less than or equal to at least one of the A_i.

However, this definition is not well-founded. The definition of the set $R_{A\to B}$ (and $R_{\{A_i\to B_i\}_{i\in I}}$) is given in terms of a set $R_{A'}$ that might not be "structurally smaller than" $R_{A\to B}$ (and $R_{\{A_i\to B_i\}_{i\in I}}$). Consider the types $A \equiv \{\}$ and $B \equiv \{A \to A\}$. Then $B \leq A$, but B is intuitively "bigger" than A, since A occurs in it. This partly explains why the $\lambda\&$-calculus is not strongly normalizing[26]. On the other hand, this observation helps finding a normalizing subsystem. A possible solution to this problem is to define a measure function *rank* from types to naturals and require that each subtyping rule for a judgment $B \leq A$ has an additional condition that $\mathit{rank}(B) \leq \mathit{rank}(A)$. The *rank* function should have the property that if B is a proper subexpression of A, then $\mathit{rank}(B) < \mathit{rank}(A)$.

To adapt the above technique to $\lambda\Pi^{\&}$, we define the *rank* function as *pSize* which is a partial approximation of the size of a type, where the information relevant to type family is ignored. More precisely, the measure *pSize* is defined as follows:

$$
\begin{array}{lcl}
\mathit{pSize}(\alpha) & = & 0 \\
\mathit{pSize}(AM) & = & \mathit{pSize}(A) \\
\mathit{pSize}(\pi x{:}A.B) & = & \mathit{pSize}(A) + \mathit{pSize}(B) + 1 \\
\mathit{pSize}(\Lambda x{:}A.B) & = & \mathit{pSize}(B) \\
\mathit{pSize}(\{\pi x{:}A_i.B_i\}_{i\in I}) & = & \max_{i\in I}\{\mathit{pSize}(\pi x{:}A_i.B_i)\} + 1
\end{array}
$$

Obviously, if B is a proper subexpression of a type A, then $\mathit{pSize}(B) < \mathit{pSize}(A)$. Furthermore, for any term N, $\mathit{pSize}(A[x := N]) = \mathit{pSize}(A)$.

DEFINITION 5.1 ($\lambda\Pi^{-}_{\&}$ system). *The $\lambda\Pi^{-}_{\&}$ system is the subsystem of $\lambda\Pi^{\&}$ where each subtyping rule for a judgment $\Gamma \vdash B \leq A$ has the additional condition pSize$(B) \leq$ pSize(A).*

The main result of this section is that terms in $\lambda\Pi^{-}_{\&}$ are strongly normalizing.

Example *Consider again the types $A \equiv \{\}$ and $B \equiv \{A \to A\}$. Note that $B \leq A$ holds in $\lambda\Pi^{\&}$ but it does not hold in $\lambda\Pi^{-}_{\&}$ since pSize$(A) =$ pSize$(\{\}) = 1 < 4 =$ pSize$(\{\ \{\} \to \{\}\ \}) =$ pSize(B).*
Intuitively, it is clear that in a normalizing calculus $B \leq A$ must not hold. Otherwise $\lambda x{:}B.x{\bullet}x$ would be well-typed (with type $B \to A$) and from such a term it would not be too difficult to derive a non-normalizing term. $\Box$

5.1. Typed inductive property

In the Tait method, we need to prove that every well-typed term belongs to the reducible set. Such a proof is difficult at the presence of overloaded types. To make things simpler, in [CGL95], an intermediate notion of set of terms being "typed inductive" is introduced.

The main steps in the proof strong normalization of $\lambda\&^{-}$-calculus (and of $\lambda\Pi^{-}_{\&}$) are:

1. Define when a set $\mathscr{S}$ is *typed inductive*,
2. Define $\mathscr{S}^\star$ the *application closure* of the typed inductive set,

[26] For example, note that the type $B \equiv \{\ \{\} \to \{\}\ \}$ is trivially well-formed. Since $B \leq \{\}$, then the application of a term of type B to itself is well-typed. Once we are able to type auto-application it is then very easy to define non-normalizing terms or fix-point combinators (see [CGL95] for details).

3. Show that $M \in \mathscr{S} \Leftrightarrow M \in \mathscr{S}^\star \Leftrightarrow M$ is well-typed,
4. Show that the set of (well-typed) strongly normalizing terms is typed inductive.

In order to ensure that the typed-inductive property and its application closure are well-defined, we need an inductive measure that satisfies the following properties:

$$\begin{array}{lll} \textit{Weight}_\Gamma(AM_1..M_n) & < \textit{Weight}_\Gamma(\alpha M_1..M_n) & \textit{if } \alpha \leq A \in \Gamma \\ \textit{Weight}_\Gamma(B[x := M_1]M_2..M_n) & < \textit{Weight}_\Gamma((\Lambda x{:}A.B)M_1..M_n) & \end{array}$$

The *Weight*$_\Gamma$ should be based on both β_2-reduction and Γ-reduction. This suggests to include the maximal number of steps of $\beta_2\Gamma$ reduction in the induction measure. Observe that we need only to consider the reductions such that the redex is at the head of a term. So we define the *head* Γ*-reduction*, denoted by $\rightarrow_{h\Gamma}$, and the *head* β_2*-reduction*, denoted by $\rightarrow_{h\beta_2}$, as follows:

$$\begin{array}{lll} \alpha M_1..M_n & \rightarrow_{h\Gamma} & \Gamma(\alpha)M_1..M_n \\ (\Lambda x{:}A.B)M_1..M_n & \rightarrow_{h\beta_2} & B[x := M_1]M_2..M_n \end{array}$$

The *head* $\beta_2\Gamma$*-reduction*, denoted by $\rightarrow_{h\beta 2\Gamma}$, is defined as the reflexive and transitive closure of $\rightarrow_{h\Gamma} \cup \rightarrow_{h\beta_2}$. Since $\beta_2\Gamma$-reduction is normalizing, so does the head $\beta_2\Gamma$-reduction. Given a pre-type A and a context Γ, we define the measure *MaxRed*$_\Gamma(A)$ as the maximal length of head $\beta_2\Gamma$-reduction from A. Note that a head $\beta_2\Gamma$-redex for A, if it exists, is always A itself, especially there are neither $h\beta_2\Gamma$-reduction in a type label, such as B in $\Lambda x{:}B.C$, nor in a term. Hence, *MaxRed*$_\Gamma$ is invariant under substitutions on term variables: *MaxRed*$_\Gamma(A[x := N]) =$ *MaxRed*$_\Gamma(A)$.

Furthermore, the measure *pSize* should be taken into account. So we define *Weight*$_\Gamma$ as the lexicographical order of *MaxRed*, defined in Section 4.4, and *pSize*:

$$\textit{Weight}_\Gamma(A) = \ <\textit{MaxRed}_\Gamma(A), \textit{pSize}(A)>$$

Observe that *MaxRed*$_\Gamma(A)$, and so *Weight*$_\Gamma(A)$, depends only on the subtyping declarations of Γ. Therefore, we have, for example, *Weight*$_{\Gamma,x:B}(A) =$ *Weight*$_\Gamma(A)$.

The following lemma shows that *Weight*$_\Gamma$ can be used as an induction measure in the definitions and proofs concerning type structure.

LEMMA 5.2 (Properties of *Weight*$_\Gamma$). *Suppose that all types in the following statements are well-formed under the context* Γ.

1.Weight$_\Gamma(A[x := N]) =$ *Weight*$_\Gamma(A)$
2.$\alpha \leq A \in \Gamma \Rightarrow$ *Weight*$_\Gamma(AM_1..M_n) <$ *Weight*$_\Gamma(\alpha M_1..M_n)$ *for any set* $M_1, .., M_n$.
3.Weight$_\Gamma(B[x := M_1]M_2..M_n) <$ *Weight*$_\Gamma((\Lambda x{:}A.B)M_1..M_n)$
4.Weight$_\Gamma(A)$, *Weight*$_\Gamma(B) <$ *Weight*$_\Gamma(\pi x{:}A.B)$
5.Weight$_\Gamma(B[x := N]) <$ *Weight*$_\Gamma(\pi x{:}A.B)$

Proof. 1) By the definition of *Weight*$_\Gamma(A)$ and the properties that *MaxRed*$_\Gamma(A[x := N]) =$ *MaxRed*$_\Gamma(A)$ and *pSize*$(A[x := N]) =$ *pSize*(A); 2) By Γ-reduction; 3) By β_2-reduction; 4) By *pSize*. Note that *Weight*$_\Gamma(B) =$ *Weight*$_{\Gamma,x:A}(B)$; 5) By 1) and 4). $\square$

NOTATION 5.3 ($M \circ N$). *We use* $M \circ N$ *to denote either* $M \cdot N$ *or* $M \bullet N$ *(according to the type of* M*).* $N \circ \vec{M}$ *denotes* $N \circ M_1 \circ ... \circ M_n$ *for* $n \geq 0$.

One of the differences between $\lambda\Pi^{\&}$ and $\lambda\&$ is the explicit use of the context Γ in the typing rules. Hence, a typed inductive set must be indexed on both type and context: $\mathscr{S}_{<\Gamma,A>}$. The following definition of a basic set characterizes context related properties that a typed inductive set must satisfy. The definition for typed inductive set is based on the notion of basic set.

DEFINITION 5.4 (Basic set). *A family $\mathscr{S}$ of sets of terms $\{\mathscr{S}_{<\Gamma,A>}\}$, indexed over well-formed context Γ and type A, is a* basic set *if*

1. $M \in \mathscr{S}_{<\Gamma,A>} \ \Rightarrow\ \Gamma \vdash M : A$,
2. $\Gamma \vdash A : \star$,
3. $\Gamma' \vdash \star\ \wedge\ \Gamma \subseteq \Gamma'\ \wedge\ \Gamma \vdash A : \star\ \Rightarrow\ \mathscr{S}_{<\Gamma,A>} \subseteq \mathscr{S}_{<\Gamma',A>}$,
4. $\Gamma \vdash A, A' : \star\ \wedge\ \Gamma \vdash A \leq A'\ \Rightarrow\ \mathscr{S}_{<\Gamma,A>} \subseteq \mathscr{S}_{<\Gamma,A'>}$.

where $\Gamma \subseteq \Gamma'$ *means that* Γ' *is an extension of* Γ', *that is,* $\Gamma' = \Gamma, \Gamma''$ *for some* Γ''.

An example of a basic set $\mathscr{S}_{<\Gamma,A>}$ is the set of normalized terms of type A under the context Γ.

We write $M \in \mathscr{S}$ if there exists Γ, A such that $M \in \mathscr{S}_{<\Gamma,A>}$ and $\vec{M} \in \mathscr{S}$ if $M_1 \in \mathscr{S}, ..., M_n \in \mathscr{S}$. When Γ is clear from the context, we may omit the label Γ in $\mathscr{S}_{<\Gamma,A>}$, and just write $\mathscr{S}_A$.

A special case of the fourth condition of basic set is when $A =_\beta A'$, we have

FACT 5.5. $\Gamma \vdash A, A' : \star\ \wedge\ A =_\beta A'\ \Rightarrow\ \mathscr{S}_{<\Gamma,A>} = \mathscr{S}_{<\Gamma,A'>}$,

Proof. By the admissibility of reflexivity. $\square$

Now, we need to adapt the notation $\in_{if}\ \mathscr{S}_A$ from [CGL95]. Here we need to add the context to the subscript. Intuitively, $M \in_{if} \mathscr{S}_{<\Gamma,A>}$ means that "if M has type A, then M belongs to $\mathscr{S}_{<\Gamma,A>}$".

NOTATION 5.6 ($\in_{if}\ \mathscr{S}_{<\Gamma,A>}$).

$$M \in_{if} \mathscr{S}_{<\Gamma,A>}\ \Leftrightarrow\ (\Gamma \vdash M : A\ \Rightarrow\ M \in \mathscr{S}_{<\Gamma,A>})$$

The next definition introduces the notion $M^{\Gamma,A} \circ \vec{N} \in_{if} \mathscr{S}$. Intuitively, it means that for M with type A under the environment Γ, if $M \circ \vec{N}$ is well-typed, then it belongs to $\mathscr{S}$. M may have some other types, say B, but the condition here does not require that $M \circ \vec{N}$ is well-typed when M is considered as a term in type B. More precisely, we have

DEFINITION 5.7 ($M^{\Gamma,A} \circ \vec{N} \in_{if} \mathscr{S}$). *Suppose that $\mathscr{S}$ is a basic set, $\vec{N} \in \mathscr{S}$ and that $\Gamma \vdash M : A$. The relation $M^{\Gamma,A} \circ \vec{N} \in_{if} \mathscr{S}$ is defined as follows,*

1. $M^{\Gamma,A} \in_{if} \mathscr{S}\ \Leftrightarrow\ M \in_{if} \mathscr{S}_{<\Gamma,A>}$;
2. $M^{\Gamma,\alpha M_1..M_n} \circ N \circ \vec{N'} \in_{if} \mathscr{S} \Leftrightarrow M^{\Gamma,A M_1..M_n} \circ N \circ \vec{N'} \in_{if} \mathscr{S}$ *for* $\alpha \leq A : K \in \Gamma$;
3. $M^{\Gamma,(\Lambda x:A.B) M_1..M_n} \circ N \circ \vec{N'} \in_{if} \mathscr{S}\ \Leftrightarrow\ M^{\Gamma,B[x:=M_1]M_2..M_n} \circ N \circ \vec{N'} \in_{if} \mathscr{S}$;
4. $M^{\Gamma,\pi x:A.B} \circ N \circ \vec{N'} \in_{if} \mathscr{S}\ \Leftrightarrow$
$\quad (N \in \mathscr{S}_{<\Gamma,A>}\ \wedge\ MN \in_{if} \mathscr{S}_{<\Gamma,B[x:=N]>}\ \wedge\ (MN)^{\Gamma,B[x:=N]} \circ \vec{N'} \in_{if} \mathscr{S})$;
5. $M^{\Gamma,\{\pi x:A_i.B_i\}_{i\in I}} \circ N \circ \vec{N'} \in_{if} \mathscr{S}\ \Leftrightarrow\ (\exists i \in I.$
$\quad N \in \mathscr{S}_{<\Gamma,A_i>}\ \wedge\ M \bullet N \in_{if} \mathscr{S}_{<\Gamma,B_i[x:=N]>}\ \wedge\ (M \bullet N)^{\Gamma,B_i[x:=N]} \circ \vec{N'} \in_{if} \mathscr{S})$;

By *Weight*$_\Gamma(A)$, the relation $M^{\Gamma,A} \circ \vec{N} \in_{if} \mathscr{S}$ is well-defined.

Note that the case $M^{\Gamma,\alpha M_1..M_n} \circ \vec{N} \in_{if} \mathscr{S}$ where $\alpha : K \in \Gamma$ is covered in the first case. For such M, $MN_1..N_n$ with $n \geq 1$ will never be well typed.

For convenience, the above notion is extended to the general notion $M \in_{if} \mathscr{S}$.

DEFINITION 5.8 ($M \in_{if} \mathscr{S}$). *The relation $M \in_{if} \mathscr{S}$ is defined over the structure of term M,*

1.$x \in_{if} \mathscr{S} \;\Leftrightarrow\; (\exists \Gamma, A.\; x \in_{if} \mathscr{S}_{<\Gamma,A>})$
2.$\epsilon \in_{if} \mathscr{S} \;\Leftrightarrow\; True$
3.$\lambda x : C.N \in_{if} \mathscr{S} \Leftrightarrow (\exists \Gamma, A. \Gamma \vdash \lambda x{:}C.N : \pi x{:}C.A \Rightarrow \lambda x{:}C.N \in_{if} \mathscr{S}_{<\Gamma,\pi x:C.A>})$
4.$M_1 \&^W M_2 \in_{if} \mathscr{S} \;\Leftrightarrow\; \exists \Gamma\;\; M_1 \&^W M_2 \in_{if} \mathscr{S}_{<\Gamma,W>}$
5.$M \circ N \circ \vec{N}' \in_{if} \mathscr{S} \;\Leftrightarrow\; \exists \Gamma, A.\; M^{\Gamma,A} \circ N \circ \vec{N}' \in_{if} \mathscr{S}$.

With these preparations, we can now introduce the notions of typed inductive set and its application closure.

DEFINITION 5.9 (Typed Inductive Set). *Let $\mathscr{S} = \{\mathscr{S}_{<\Gamma,A>}\}$ be a basic set of $\lambda\Pi^-_{\&}$ terms and $\vec{N}$ a sequence of well-typed terms. $\mathscr{S}$ is* typed-inductive *if it satisfies the following conditions :*

$(\varepsilon)\quad \varepsilon \in \mathscr{S}_{<\Gamma,\{\}>}$

$(x)\quad \forall x \in \mathscr{S}_{<\Gamma,A>}, \vec{N} \in \mathscr{S}.\; x^{\Gamma,A} \circ \vec{N} \in_{if} \mathscr{S}$

$(\&_1)\; \forall M_1 \in \mathscr{S}_{<\Gamma,W>}, M_2 \in \mathscr{S}_{<\Gamma,\pi x:A.B>}, \vec{N} \in \mathscr{S}.$
$\quad (M_1^{\Gamma,W} \circ \vec{N} \in_{if} \mathscr{S} \wedge M_2^{\Gamma,\pi x:A.B} \circ \vec{N} \in_{if} \mathscr{S} \Rightarrow (M_1 \&^{W \cup \{\pi x:A.B\}} M_2) \circ \vec{N} \in_{if} \mathscr{S})$

$(\lambda_1)\; \forall M \in \mathscr{S}_{<\Gamma,A>}, \vec{N} \in \mathscr{S}.\;\; M[x := N]^{\Gamma,A} \circ \vec{N} \in_{if} \mathscr{S}$
$\qquad \Rightarrow\; (\lambda x{:}B.M)^{\Gamma,\pi x:B.A} \cdot N \circ \vec{N} \in_{if} \mathscr{S}$

$(\&_2)\; \forall M_1 \in \mathscr{S}_{<\Gamma,W>}, M_2 \in \mathscr{S}_{<\Gamma,\pi x:A.B>}.\;\; M_1 \&^{W \cup \{\pi x:A.B\}} M_2 \in_{if} \mathscr{S}$

$(\lambda_2)\; \forall M \in \mathscr{S}_{<(\Gamma,x:A),B>}.\;\; \lambda x{:}A.M \in_{if} \mathscr{S}_{<\Gamma,\pi x:A.B>}$

DEFINITION 5.10 (Application Closure of $\mathscr{S}$). *Let $\{\mathscr{S}_{<\Gamma,A>}\}$ be a typed-inductive set. Its* application closure *in $\lambda\Pi^-_{\&}$, denoted by $\{\mathscr{S}^\star_{<\Gamma,A>}\}$, is inductively defined on the structure of A as follows:*

<u>CASE</u> $\alpha M_1..M_n$ *There are two subcases: (1) if $\alpha : K \in \Gamma$, then $M \in \mathscr{S}^\star_{<\Gamma,\alpha M_1..M_n>} \Leftrightarrow M \in \mathscr{S}_{<\Gamma,\alpha M_1..M_n>}$. (2) if $\alpha \leq A : K \in \Gamma$, then $M \in \mathscr{S}^\star_{<\Gamma,\alpha M_1..M_n>} \Leftrightarrow M \in \mathscr{S}_{<\Gamma,\alpha M_1..M_n>} \wedge M \in \mathscr{S}^\star_{<\Gamma,AM_1..M_n>}$.*

<u>CASE</u> $(\Lambda x{:}A.B)M_1..M_n$: $M \in \mathscr{S}^\star_{<\Gamma,(\Lambda x:A.B)M_1..M_n>} \Leftrightarrow M \in \mathscr{S}_{<\Gamma,(\Lambda x:A.B)M_1..M_n>} \wedge M \in \mathscr{S}^\star_{<\Gamma,B[x:=M_1]M_2..M_n>}$.

<u>CASE</u> $\pi x{:}A.B$: $M \in \mathscr{S}^\star_{<\Gamma,\pi x:A.B>} \Leftrightarrow M \in \mathscr{S}_{<\Gamma,\pi x:A.B>} \wedge (\forall \Gamma', N. \Gamma' \vdash \star \wedge \Gamma \subseteq \Gamma' \wedge N \in \mathscr{S}^\star_{<\Gamma',A>} \Rightarrow MN \in \mathscr{S}^\star_{<\Gamma',B[x:=N]>})$.

<u>CASE</u> $\{\pi x{:}A_i.B_i\}_{i\in I}$: $M \in \mathscr{S}^\star_{<\Gamma,\{\pi x:A_i.B_i\}_{i\in I}>} \Leftrightarrow M \in \mathscr{S}_{<\Gamma,\{\pi x:A_i.B_i\}_{i\in I}>} \wedge (\forall i{\in}I. \forall \Gamma', N. \Gamma' \vdash \star \wedge \Gamma \subseteq \Gamma' \wedge N \in \mathscr{S}^\star_{<\Gamma',A_i>} \Rightarrow M \bullet N \in \mathscr{S}^\star_{<\Gamma',B_i[x:=N]>})$.

This definition is well-formed.

LEMMA 5.11 (Well-definedness of $\mathscr{S}^\star_{<\Gamma,A>}$). *The set $\mathscr{S}^\star_{<\Gamma,A>}$ is well-defined on each well-formed type A.*

Proof. Induction on *Weight*$_\Gamma(A)$. $\Box$

The notion of application closure has a simpler presentation:

LEMMA 5.12 (Equivalent presentation of application closure). *$M \in \mathscr{S}^\star_{<\Gamma,A>}$ if and only if $M \in \mathscr{S}_{<\Gamma,A>} \wedge (\forall\Gamma', N.\, \Gamma' \vdash \star \wedge \Gamma \subseteq \Gamma' \wedge N \in \mathscr{S}^\star_{<\Gamma',B>} \Rightarrow M^{\Gamma,A} \circ N \in_{if} \mathscr{S}^\star)$*

Proof. Induction on *Weight*$_\Gamma(A)$. $\Box$

5.2. The application closure

A typed inductive set $\mathscr{S}$ is a basic set, so $\mathscr{S}$ has all four properties of Definition 5.4. The application closure $\mathscr{S}^\star$ is a "subset" of $\mathscr{S}$ in the sense that $\mathscr{S}^\star_{<\Gamma,A>} \subseteq \mathscr{S}_{<\Gamma,A>}$ for each context Γ and type A. Evidently, not all "subsets" of $\mathscr{S}$ enjoy the nice properties of basic sets. But we can show that $\mathscr{S}^\star$ is still a basic set. Actually we need only to verify that the last two conditions of basic set hold for $\mathscr{S}^\star$.

First we show that $\mathscr{S}^\star_{<\Gamma,A>}$ enjoys the third property of basic sets.

LEMMA 5.13 (Invariance of $\mathscr{S}^\star$ under context extension).
$$\Gamma' \vdash \star \wedge \Gamma \subseteq \Gamma' \wedge \Gamma \vdash A : \star \Rightarrow \mathscr{S}^\star_{<\Gamma,A>} \subseteq \mathscr{S}^\star_{<\Gamma',A>}$$

Proof. By the definition of $\mathscr{S}^\star$, $\mathscr{S}$ is typed inductive, thus a basic set. Therefore,

$$\Gamma' \vdash \star \wedge \Gamma \subseteq \Gamma' \wedge \Gamma \vdash A : \star \Rightarrow \mathscr{S}_{<\Gamma,A>} \subseteq \mathscr{S}_{<\Gamma',A>}$$

The proof proceeds by induction on *Weight*$_\Gamma(A)$. $\Box$

Given a term $M \in \mathscr{S}^\star_{<\Gamma,\pi x:A.B>}$, we want to show that, for any term $N \in \mathscr{S}^\star_{<\Gamma,A>}$, we have $MN \in \mathscr{S}^\star_{<\Gamma,B[x:=N]>}$. Due to subtyping, it may happen that M will be applied to an argument $N \in \mathscr{S}^\star_{<\Gamma,A'>}$ where $A' \leq A$. Hence, we want to show that

$$\Gamma \vdash A' \leq A \Rightarrow \mathscr{S}^\star_{<\Gamma,A'>} \subseteq \mathscr{S}^\star_{<\Gamma,A>}$$

A special case of this property is when A' and A are β-convertible:

$$A =_\beta A' \Rightarrow \mathscr{S}^\star_{<\Gamma,A>} = \mathscr{S}^\star_{<\Gamma,A'>}$$

To prove this property, we first study a special case where the β conversion is restricted to head β_2 reduction, which has been defined as (see Section 5.1)

$$(\Lambda x{:}A.B)M_1...M_n \rightarrow_{h\beta_2} B[x := M_1]M_2..M_n$$

$\mathscr{S}^\star$ is invariant under head β_2 reduction.

LEMMA 5.14 ($\mathscr{S}^\star$ and head β_2-reduction).
$$\Gamma \vdash A : \star \wedge A \rightarrow_{h\beta_2} A' \Rightarrow \mathscr{S}^\star_{<\Gamma,A>} = \mathscr{S}^\star_{<\Gamma,A'>}$$

Let $A^{h\beta_2}$ denote the head normal form of $h\beta_2$ reduction.

COROLLARY 5.15 (β_2 hnf and $\mathscr{S}^\star$). $\Gamma \vdash A : \star \Rightarrow \mathscr{S}^\star_{<\Gamma,A>} = \mathscr{S}^\star_{<\Gamma,A^{h\beta_2}>}$

Now we show that $\mathscr{S}^\star$ is invariant under β conversion.

LEMMA 5.16 ($\mathscr{S}^\star$ type conversion).
$$\Gamma \vdash A, A' : \star \ \wedge \ A =_\beta A' \ \Rightarrow \ \mathscr{S}^\star_{<\Gamma,A>} = \mathscr{S}^\star_{<\Gamma,A'>}$$

Proof. First note that $\Gamma \vdash A, A' : \star \ \wedge \ A =_\beta A' \ \Rightarrow \ \mathscr{S}_{<\Gamma,A>} = \mathscr{S}_{<\Gamma,A'>}$ The proof proceeds by induction on *Weight*$_\Gamma(A)$. □

Now we can show the main result of this section: $\mathscr{S}^\star_{<\Gamma,A>}$ is monotonic with respect to subtyping.

LEMMA 5.17 (Subtyping implies application closure containment).
$$\Gamma \vdash A \leq B \ \wedge \ \Gamma \vdash A, B : \star \Rightarrow \ \mathscr{S}^\star_{<\Gamma,A>} \subseteq \mathscr{S}^\star_{<\Gamma,B>}$$

Proof. Since typed inductive set is basic set, we have
$$\Gamma \vdash A \leq B \ \Rightarrow \ \mathscr{S}_{<\Gamma,A>} \subseteq \mathscr{S}_{<\Gamma,B>}$$
Given A, B satisfying the condition of the lemma and $M \in \mathscr{S}^\star_{<\Gamma,A>}$, we need to show that $M \in \mathscr{S}^\star_{<\Gamma,B>}$. Note that, we have $M \in \mathscr{S}_{<\Gamma,B>}$.

The proof proceeds by induction on the derivation depth of *Weight*$_\Gamma(A)$ + *Weight*$_\Gamma(B)$. Note that for any Γ' such that $\Gamma' \vdash \star \ \wedge \ \Gamma \subseteq \Gamma'$,

$$\textit{Weight}_{\Gamma'}(A) + \textit{Weight}_{\Gamma'}(B) \ = \ \textit{Weight}_{\Gamma}(A) + \textit{Weight}_{\Gamma}(B)$$

We proceed according to the last rule used to derive $\Gamma \vdash A \leq B$.

<u>CASE</u> (S-π). Assume that the last step of the derivation for $\Gamma \vdash A \leq B$ is

$$\frac{\Gamma \vdash C \leq E \quad \Gamma, x : C \vdash F \leq D}{\Gamma \vdash \pi x{:}E.F \leq \pi x{:}C.D} \ \text{S-}\pi$$

where $A \equiv \pi x{:}E.F, B \equiv \pi x{:}C.D$. Let $M \in \mathscr{S}^\star_{<\Gamma,\pi x:E.F>}$; we need to show that

$$\forall \Gamma', N. \ \ \Gamma' \vdash \star \ \wedge \ \Gamma \subseteq \Gamma' \ \wedge \ N \in \mathscr{S}^\star_{<\Gamma',C>} \ \Rightarrow \ MN \in \mathscr{S}^\star_{<\Gamma',D[x:=N]>}$$

Let Γ' be such that $\Gamma' \vdash \star \ \wedge \ \Gamma \subseteq \Gamma'$. The proof proceeds as follows,

$$\begin{array}{lll}
N \in \mathscr{S}^\star_{<\Gamma',C>} & \Rightarrow \ N \in \mathscr{S}_{<\Gamma',C>} & \\
& \Rightarrow \ \Gamma' \vdash N : C & \\
\Gamma' \vdash C \leq E & \Rightarrow \ \mathscr{S}^\star_{<\Gamma',C>} \subseteq \mathscr{S}^\star_{<\Gamma',E>} & \textit{induction hypothesis} \ \ (IH) \\
\Gamma', x : C \vdash F \leq D & \Rightarrow \ \Gamma' \vdash F[x := N] \leq D[x := N] & \\
& \ \ \ \ \wedge \ \textit{pSize}(F[x := N]) = \textit{pSize}(F) < \textit{pSize}(\pi x{:}E.F) & \\
& \ \ \ \ \wedge \ \textit{pSize}(D[x := N]) = \textit{pSize}(D) < \textit{pSize}(\pi x{:}C.D) & \\
& \Rightarrow \ \mathscr{S}^\star_{<\Gamma',F[x:=N]>} \subseteq \mathscr{S}^\star_{<\Gamma',D[x:=N]>} & IH \\
\mathscr{S}^\star_{<\Gamma',C>} \subseteq \mathscr{S}^\star_{<\Gamma',E>} & \Rightarrow \ N \in \mathscr{S}^\star_{<\Gamma',E>} & \\
M \in \mathscr{S}^\star_{<\Gamma,\pi x:E.F>} & \Rightarrow \ M \in \mathscr{S}^\star_{<\Gamma',\pi x:E.F>} & \\
& \Rightarrow \ MN \in \mathscr{S}^\star_{<\Gamma',F[x:=N]>} & \\
& \Rightarrow \ MN \in \mathscr{S}^\star_{<\Gamma',D[x:=N]>} & \\
& \Rightarrow \ \mathscr{S}^\star_{<\Gamma',\pi x:E.F>} \subseteq \mathscr{S}^\star_{<\Gamma',\pi x:C.D>} &
\end{array}$$

CASE (S-ApR). Suppose that the last step of the derivation for $\Gamma \vdash A \leq B$ is

$$\frac{M_1 =_\beta M_1' \cdots M_n =_\beta M_n'}{\Gamma \vdash \alpha M_1..M_n \leq \alpha M_1'..M_n'} \text{ S-ApR}$$

The result follows from the Lemma 5.16.

CASE (S-ApT). Suppose the last step of the derivation for $\Gamma \vdash A \leq B$ is

$$\frac{\Gamma \vdash \Gamma(\alpha)M_1..M_n \leq B}{\Gamma \vdash \alpha M_1..M_n \leq B} \text{ S-ApT}$$

Then

$$\begin{array}{rcll} \mathscr{S}^\star_{<\Gamma,\alpha M_1..M_n>} & = & \mathscr{S}^\star_{<\Gamma,\Gamma(\alpha)M_1..M_n>} & \textit{definition of } \mathscr{S}^\star \\ & \subseteq & \mathscr{S}^\star_{<\Gamma,B>} & IH \end{array}$$

CASES (S-ApSL) and (S-ApSR). Similar.

CASE (S-OVER). Suppose that the last step of the derivation of $\Gamma \vdash A \leq B$ is

$$\frac{\forall j \in J \exists i \in I \;\; \Gamma \vdash \pi x{:}A_i.B_i \leq \pi x{:}C_j.D_j}{\Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\in I} \leq \{\pi x{:}C_j.D_j\}_{j\in J}} \text{ S-OVER}$$

where $A \equiv \{\pi x{:}A_i.B_i\}_{i\in I}, B \equiv \{\pi x{:}C_j.D_j\}_{j\in J}$.

Assume that $M \in \mathscr{S}^\star_{<\Gamma,\{\pi x:A_i.B_i\}_{i\in I}>}$ and $N \in \mathscr{S}^\star_{<\Gamma,C_j>}$. Let Γ' be such that $\Gamma' \vdash \star \;\wedge\; \Gamma \subseteq \Gamma'$, we need to prove that $M \bullet N \in \mathscr{S}^\star_{<\Gamma',D_j[x:=N]>}$. We proceed as follows:

$$\begin{array}{lll} \multicolumn{2}{l}{\Gamma' \vdash \{\pi x{:}A_i.B_i\}_{i\in I} \leq \{\pi x{:}C_j.D_j\}_{j\in J}} & \\ \Rightarrow & \exists h \in I \;\; \Gamma' \vdash \pi x{:}A_h.B_h \leq \pi x{:}C_j.D_j & \textit{by definition} \\ & \wedge\, \textit{pSize}(\pi x{:}A_h.B_h) < \textit{pSize}(A) \;\wedge\; \textit{pSize}(\pi x{:}C_j.D_j) < \textit{pSize}(B) & \\ \Rightarrow & \Gamma' \vdash C_j \leq A_h \;\wedge\; \Gamma', x : C_j \vdash B_h \leq D_j & \text{S-}\pi \\ & \wedge\, \textit{pSize}(C_j) < \textit{pSize}(\pi x{:}C_j.D_j) \;\wedge\; \textit{pSize}(A_h) < \textit{pSize}(\pi x{:}A_h.B_h) & \\ \Rightarrow & \mathscr{S}^\star_{<\Gamma',C_j>} \subseteq \mathscr{S}^\star_{<\Gamma',A_h>} & IH \\ \Rightarrow & N \in \mathscr{S}^\star_{<\Gamma',A_h>} & \\ \Rightarrow & M \bullet N \in \mathscr{S}^\star_{<\Gamma',B_h[x:=N]>} & \textit{by def. of } \mathscr{S}^\star \end{array}$$

On the other hand,

$$\begin{array}{lll} \multicolumn{2}{l}{\Gamma', x : C_j \vdash B_h \leq D_j} & \\ \Rightarrow & \Gamma' \vdash B_h[x := N] \leq D_j[x := N] \;\wedge & \\ & \textit{pSize}(B_h[x := N]) = \textit{pSize}(B_h) < \textit{pSize}(\pi x{:}A_h.B_h) < \textit{pSize}(A) \,\wedge & \\ & \textit{pSize}(D_j[x := N]) = \textit{pSize}(D_j) < \textit{pSize}(\pi x{:}C_j.D_j) < \textit{pSize}(B) & \\ \Rightarrow & \mathscr{S}^\star_{<\Gamma',B_h[x:=N]>} \subseteq \mathscr{S}^\star_{<\Gamma',D_j[x:=N]>} & IH \\ \Rightarrow & M \bullet N \in \mathscr{S}^\star_{<\Gamma',D_j[x:=N]>} & \\ \Rightarrow & M \in \mathscr{S}^\star_{<\Gamma',\{\pi x:C_j.D_j\}_{j\in J}>} & \textit{by def. of } \mathscr{S}^\star \\ \Rightarrow & \mathscr{S}^\star_{<\Gamma',\{\pi x:A_i.B_i\}_{i\in I}>} \subseteq \mathscr{S}^\star_{<\Gamma',\{\pi x:C_j.D_j\}_{j\in J}>} & \end{array}$$

□

COROLLARY 5.18.

$$\Gamma \vdash A \leq B \;\wedge\; \Gamma \vdash B \leq A \;\wedge\; \Gamma \vdash A, B : \star \;\Rightarrow\; \mathscr{S}^\star_{<\Gamma,A>} = \mathscr{S}^\star_{<\Gamma,B>}$$

5.3. Relation between $\mathscr{S}$ and $\mathscr{S}^\star$

In this section we study the relationship between typed-inductive set $\mathscr{S}$ and its application closure $\mathscr{S}^\star$. The main result is Corollary 5.22, that states that every typed inductive set contains all the well typed terms. Its proof relies on the following lemma. In the following, $N \in \mathscr{S}^\star$ will mean that $N \in \mathscr{S}^\star_{<\Gamma,A>}$ for some type A and environment Γ.

LEMMA 5.19 ($\mathscr{S}_{if}$ implies $\mathscr{S}^\star$).

$$M \in \mathscr{S}^\star_{<\Gamma,A>} \;\Leftrightarrow\; \Gamma \vdash M : A \;\wedge\; \forall \vec{N} \in \mathscr{S}^\star . M^{\Gamma,A} \circ \vec{N} \in_{if} \mathscr{S}$$

Proof.

(⇒) We prove the stronger property

$$M \in \mathscr{S}^\star_{<\Gamma,A>} \;\Rightarrow\; \forall \vec{N} \in \mathscr{S}^\star .\, M^{\Gamma,A} \circ \vec{N} \in_{if} \mathscr{S}^\star$$

by induction on the length of $\vec{N}$.

(⇐) When $\vec{N}$ is empty, by definition, we have

$$\Gamma \vdash M : A \;\wedge\; M^{\Gamma,A} \in_{if} \mathscr{S} \;\Rightarrow\; M \in \mathscr{S}^\star_{<\Gamma,A>}$$

The proof proceeds by induction on *Weight*$_\Gamma(A)$.

□

The following lemma shows that if $x : A \in \Gamma$, then A is a minimal type[27] for the term variable x.

LEMMA 5.20 (Minimal type for variable).

$$\Gamma_1, x : A, \Gamma_2 \vdash x : B \;\Rightarrow\; \Gamma_1, x : A, \Gamma_2 \vdash A \leq B$$

Proof. By observation of the typing rules □

Now we prove the main result of this subsection.

PROPOSITION 5.21. *For every typed-inductive set $\mathscr{S}$*

$$\Gamma \vdash M : A : \star \;\Rightarrow\; M \in \mathscr{S}^\star_{<\Gamma,A>}$$

Proof. We prove the following stronger property, given a context Γ, terms $P_1, .., P_n, M$ and types $C_1, .., C_n, B$ (with $n \geq 0$):

$$\begin{array}{l}\forall \sigma \equiv [x_1 := P_1, ..., x_n := P_n].\\ \quad (\forall i \in [1..n].\Gamma, x_1 : C_1, ..., x_{i-1} : C_{i-1} \vdash P_i : C_i \wedge P_i \in \mathscr{S}^\star_{<\Gamma, C_i[x_1 := P_1, ..., x_{i-1} := P_{i-1}]>})\\ \quad \wedge\, \Gamma, x_1 : C_1, ..., x_n : C_n \vdash M : B : \star \Rightarrow M\sigma \in \mathscr{S}^\star_{<\Gamma, B\sigma>}\end{array}$$

Let σ be the substitution satisfying the condition of the above property, $Dom(\sigma) = \{x_1, ..., x_n\}$, $\Gamma' = \Gamma, x_1 : C_1, ..., x_n : C_n$. Note that

[27] We speak of "a minimal type" rather than "the least type" since by subsumption every type B such that $\Gamma \vdash B \leq A \leq B$ is a minimal types of x, too.

1. $\Gamma'\sigma = (\Gamma, x_1 : C_1, ..., x_n : C_n)\sigma = \Gamma$ *Definition 4.5*
2. $j > i \geq 1 \Rightarrow x_j \notin Fv(C_i)$
3. $C_i[x_1 := P_1, ..., x_{i-1} := P_{i-1}] = C_i\sigma$
4. $\Gamma' \vdash A \leq B \Rightarrow \Gamma \vdash A\sigma \leq B\sigma$ *Lemma* 4.6
5. $\Gamma \vdash A\sigma \leq B\sigma : s \Rightarrow \mathscr{S}^\star_{<\Gamma, A\sigma>} \subseteq \mathscr{S}^\star_{<\Gamma, B\sigma>}$ *Lemma* 5.17
6. $\Gamma, x_1 : C_1, ..., x_{i-1} : C_{i-1} \vdash C_i : \star \wedge \Gamma \vdash C_i\sigma : \star$ *Proposition* 4.11
7. $\Gamma \vdash B\sigma : \star$ *Proposition* 4.11

Induction on the size of M. We analyse the different cases for M.

<u>Case</u> $M \equiv x$. Then, $\Gamma' \vdash x : B \Rightarrow \Gamma \vdash x\sigma : B\sigma \Rightarrow \Gamma \vdash B\sigma : \star$.
1) $x \notin Dom(\sigma)$. Then, $x\sigma = x$. Let $\vec{N} \in \mathscr{S}^\star$.

$\vec{N} \in \mathscr{S}^\star$	$\Rightarrow$	$\vec{N} \in \mathscr{S}$	*Definition* 5.10
$\Gamma' \vdash x : B$	$\Rightarrow$	$\Gamma \vdash x : B\sigma$	*substitution*
$\vec{N} \in \mathscr{S} \wedge \Gamma \vdash x : B\sigma$	$\Rightarrow$	$x^{\Gamma, B\sigma} \circ \vec{N} \in_{if} \mathscr{S}$	*Definition* 5.9
$\Gamma \vdash x : B\sigma \wedge x^{\Gamma, B\sigma} \circ \vec{N} \in_{if} \mathscr{S}$	$\Rightarrow$	$x \in \mathscr{S}^\star_{<\Gamma, B\sigma>}$	*Lemma* 5.19
$x\sigma = x$	$\Rightarrow$	$x\sigma \in \mathscr{S}^\star_{<\Gamma, B\sigma>}$	

2) $x \equiv x_i \in Dom(\sigma)$. Then,

$$x\sigma = x_i\sigma = P_i \in \mathscr{S}^\star_{<\Gamma, C_i[x_1 := P_1, ..., x_{i-1} := P_{i-1}]>} = \mathscr{S}^\star_{<\Gamma, C_i\sigma>}$$

$x_i : C_i \in \Gamma' \wedge \Gamma \vdash x_i : B$
$\Rightarrow \Gamma' \vdash C_i \leq B$ *Lemma* 5.20
$\Rightarrow \Gamma \vdash C_i\sigma \leq B\sigma$ *substitution for subtyping*

$\Gamma \vdash C_i\sigma \leq B\sigma \wedge \Gamma \vdash C_i\sigma, B\sigma : \star$
$\Rightarrow \mathscr{S}^\star_{<\Gamma, C_i\sigma>} \subseteq \mathscr{S}^\star_{<\Gamma, B\sigma>}$ *Proposition* 5.17
$\Rightarrow x\sigma \in \mathscr{S}^\star_{<\Gamma, B\sigma>}$

<u>Case</u> $M \equiv (M_1 \&^{W \cup \{\pi x:C.D\}} M_2)$. It follows from generation for typing, substitution property and the closeness of overloaded type that

$\Gamma' \vdash M_1 \&^{W \cup \{\pi x:C.D\}} M_2 : B \Rightarrow \Gamma' \vdash M_1 : W \wedge \Gamma' \vdash M_2 : \pi x{:}C.D$
$\wedge \Gamma' \vdash W \cup \{\pi x{:}C.D\} \leq B$
$\wedge \Gamma' \vdash W \cup \{\pi x{:}C.D\} : \star$
$\Rightarrow \Gamma \vdash (W \cup \{\pi x{:}C.D\})\sigma : \star$ *Prop.* 4.11

$\Gamma' \vdash W \cup \{\pi x{:}C.D\} \leq B \Rightarrow \Gamma \vdash (W \cup \{\pi x{:}C.D\})\sigma \leq B\sigma$

$\Gamma \vdash (W \cup \{\pi x{:}C.D\})\sigma \leq B\sigma : \star \Rightarrow \mathscr{S}^\star_{<\Gamma, (W \cup \{\pi x:C.D\})\sigma>} \subseteq \mathscr{S}^\star_{<\Gamma, B\sigma>}$ *Lemma* 5.17

$\Gamma' \vdash W \cup \{\pi x{:}C.D\} : \star \Rightarrow (W \cup \{\pi x{:}C.D\})\sigma = W \cup \{\pi x{:}C.D\}$
$\Rightarrow (M_1 \&^{W \cup \{\pi x:C.D\}} M_2)\sigma = (M_1\sigma) \&^{W \cup \{\pi x:C.D\}} (M_2\sigma)$

Let $\vec{N} \in \mathscr{S}^\star$,

$$\begin{array}{lll}
\Gamma' \vdash M_1 : W \;\wedge\; \Gamma' \vdash M_2 : \pi x{:}C.D & & \\
\quad \Rightarrow\; M_1\sigma \in \mathscr{S}^\star_{<\Gamma, W\sigma>} \;\wedge\; M_2\sigma \in \mathscr{S}^\star_{<\Gamma,\{\pi x:C.D\}\sigma>} & & IH \\
\quad \Rightarrow\; M_1\sigma \in \mathscr{S}^\star_{<\Gamma, W>} \;\wedge\; M_2\sigma \in \mathscr{S}^\star_{<\Gamma,\{\pi x:C.D\}>} & & W \cup \{\pi x{:}C.D\} \textit{ closed} \\
\quad \Rightarrow\; (M_1\sigma)^{\Gamma,W} \circ \vec{N} \in_{if} \mathscr{S} \;\wedge\; (M_2\sigma)^{\Gamma,\pi x:C.D} \circ \vec{N} \in_{if} \mathscr{S} & & \textit{Lemma } 5.19 \\
\quad \Rightarrow\; ((M_1 \&^{W\cup\{\pi x:C.D\}} M_2)\sigma)^{\Gamma,(W\cup\{\pi x:C.D\})\sigma} \circ \vec{N} \in_{if} \mathscr{S} & & \textit{Definition } 5.9 \\
\quad \Rightarrow\; (M_1 \&^{W\cup\{\pi x:C.D\}} M_2)\sigma \in \mathscr{S}^\star_{<\Gamma,(W\cup\{\pi x:C.D\})\sigma>} & & \textit{Lemma } 5.19 \\
\quad \Rightarrow\; (M_1 \&^{W\cup\{\pi x:C.D\}} M_2)\sigma \in \mathscr{S}^\star_{<\Gamma,B\sigma>} & &
\end{array}$$

<u>CASE</u> $M \equiv (\lambda x{:}C.M')$. Let $\vec{N} \in \mathscr{S}^\star$, first prove that $((\lambda x{:}C.M')\sigma)^{\Gamma,(\pi x:C.D)\sigma} \cdot \vec{N} \in_{if} \mathscr{S}$ for some D. Note that

$$\begin{array}{lll}
\Gamma' \vdash \lambda x{:}C.M' : B & \Rightarrow & \exists D.\, \Gamma', x : C \vdash M' : D \;\wedge\; \Gamma' \vdash \pi x{:}C.D \leq B \\
& & \quad \wedge\, \Gamma' \vdash \pi x{:}C.D : \star \\
& \Rightarrow & \Gamma \vdash (\pi x{:}C.D)\sigma : \star
\end{array}$$

$$\Gamma' \vdash \pi x{:}C.D \leq B \;\Rightarrow\; \Gamma \vdash (\pi x{:}C.D)\sigma \leq B\sigma$$

$$\Gamma \vdash (\pi x{:}C.D)\sigma \leq B\sigma : \star \quad\Rightarrow\quad \mathscr{S}^\star_{<\Gamma,(\pi x:C.D)\sigma>} \subseteq \mathscr{S}^\star_{<\Gamma,B\sigma>}$$

There are two subcases:

1) $\vec{N} \equiv N \cup \vec{N'}$:

$$\begin{array}{llll}
\Gamma', x : C \vdash M' : D & \Rightarrow & (M'\sigma[x := N]) \in \mathscr{S}^\star_{<\Gamma,D\sigma[x:=N]>} & IH \\
& \Rightarrow & (M'\sigma[x := N])^{\Gamma,D\sigma[x:=N]} \circ \vec{N'} \in_{if} \mathscr{S} & \textit{Lemma } 5.19 \\
& \Rightarrow & (\lambda x{:}C\sigma.M'\sigma)^{\Gamma,\pi x:C\sigma.D\sigma} N \circ \vec{N'} \in_{if} \mathscr{S} & \textit{Definition } 5.9\ (\lambda_1)
\end{array}$$

2) $\vec{N} \equiv \emptyset$:

$$\begin{array}{llll}
\Gamma', x : C \vdash M' : D & \Rightarrow & M'\sigma \in \mathscr{S}^\star_{<(\Gamma,x:C\sigma),D\sigma>} & IH \\
& \Rightarrow & M'\sigma \in_{if} \mathscr{S}_{<(\Gamma,x:C\sigma),D\sigma>} & \\
& \Rightarrow & \lambda x{:}C\sigma.M'\sigma \in_{if} \mathscr{S}_{<\Gamma,\pi x:C\sigma.D\sigma>} & \textit{Definition } 5.9 \\
& \Rightarrow & ((\lambda x{:}C.M')\sigma)^{\Gamma,\pi x:C\sigma.D\sigma} \in_{if} \mathscr{S} &
\end{array}$$

We conclude that $\forall \vec{N}.\ ((\lambda x{:}C.M')\sigma)^{\Gamma,(\pi x:C.D)\sigma} \vec{N} \in_{if} \mathscr{S}$. Furthermore,

$$\begin{array}{ll}
\Gamma \vdash (\lambda x{:}C.M')\sigma : (\pi x{:}C.D)\sigma \;\wedge\; ((\lambda x{:}C.M')\sigma)^{\Gamma,(\pi x:C.D)\sigma} \vec{N} \in_{if} \mathscr{S} & \\
\quad \Rightarrow\; (\lambda x{:}C.M')\sigma \in \mathscr{S}^\star_{<\Gamma,(\pi x:C.D)\sigma>} & \textit{Lemma } 5.19 \\
\quad \Rightarrow\; (\lambda x{:}C.M')\sigma \in \mathscr{S}^\star_{<\Gamma,B\sigma>} &
\end{array}$$

<u>CASE</u> $M \equiv (M' \bullet N)$. Then,

$$\begin{array}{lll}
\multicolumn{3}{l}{\Gamma' \vdash M' \bullet N : B} \\
\Rightarrow & \Gamma' \vdash M' : \{\pi y_i{:}A_i.B_i\}_{i\in I} \;\wedge\; \Gamma' \vdash N : A_i \;\wedge\; \Gamma' \vdash B_i[y_i := N] \leq B & \\
& \quad \wedge\, \Gamma \vdash (B_i[y_i := N])\sigma : \star & \\
\Rightarrow & M'\sigma \in \mathscr{S}^\star_{<\Gamma,(\{\pi y_i:A_i.B_i\}_{i\in I})\sigma>} \wedge N\sigma \in \mathscr{S}^\star_{<\Gamma,A_i\sigma>} \wedge \Gamma \vdash (B_i[y_i := N])\sigma \leq B\sigma & IH \\
\Rightarrow & M'\sigma \in \mathscr{S}^\star_{<\Gamma,\{\pi y_i:A_i.B_i\}_{i\in I}>} \wedge N\sigma \in \mathscr{S}^\star_{<\Gamma,A_i\sigma>} \wedge \Gamma \vdash B_i[y_i := N\sigma] \leq B\sigma & \\
\Rightarrow & (M'\sigma)^{\Gamma,\{\pi y_i:A_i.B_i\}_{i\in I}} \bullet (N\sigma) \in \mathscr{S}^\star_{<\Gamma,B_i[y_i:=N\sigma]>} \subseteq \mathscr{S}^\star_{<\Gamma,B\sigma>} & \textit{Definition } 5.10 \\
\equiv & (M' \bullet N)\sigma \in \mathscr{S}^\star_{<\Gamma,B\sigma>} &
\end{array}$$

<u>CASE</u> $M \equiv (M'N)$. Similar. $\square$

COROLLARY 5.22. *If S is a typed inductive set, then*

$$M \in \mathscr{S}^\star_{<\Gamma,A>} \Leftrightarrow M \in \mathscr{S}_{<\Gamma,A>} \Leftrightarrow \Gamma \vdash M : A$$

Proof.

$$\begin{array}{lll} M \in \mathscr{S}^*_{<\Gamma,A>} & \Rightarrow\ M \in \mathscr{S}_{<\Gamma,A>} & \textit{by definition of } \mathscr{S}^* \\ M \in \mathscr{S}_{<\Gamma,A>} & \Rightarrow\ \Gamma \vdash M : A & \textit{by definition of } \mathscr{S} \\ \Gamma \vdash M : A & \Rightarrow\ M \in \mathscr{S}^*_{<\Gamma,A>} & \textit{By Proposition } 5.21 \end{array}$$

□

5.4. Strong normalization is typed-inductive

Now we can prove that in $\lambda\Pi^-_\&$ strong normalization is a typed-inductive property, and therefore well-typed terms in $\lambda\Pi^-_\&$ are strongly normalizing.

Define $\mathscr{SN}_{<\Gamma,A>} = \{M \mid \Gamma \vdash M : A\ \wedge\ M \in \lambda\Pi^-_\&\ \wedge\ M \text{ is strongly normalizing}\}$ and $\mathscr{SN} = \{\mathscr{SN}_{<\Gamma,A>}\}$.

PROPOSITION 5.23 ($\mathscr{SN}$ is typed-inductive). *$\mathscr{SN}$ is typed-inductive.*

Proof. First we have to verify that $\mathscr{SN}$ is a basic set. The first three conditions of the definition of a basic set (Definition 5.4) are straightforward. For the fourth condition just note that by the subsumption rule if $\Gamma \vdash M : A$ and $\Gamma \vdash A \leq A'$, then $\Gamma \vdash M : A'$.

Then we need to show that $\mathscr{SN}$ satisfies the conditions (ε), (x), $(\&_1)$, (λ_1), $(\&_2)$ and (λ_2) of the definition of typed-inductive set. We analyse the case $(\&_1)$; others are straightforward or similar.

Assume $M_1 \in \mathscr{SN}_W, M_2 \in \mathscr{S}_{<\Gamma,\pi x:A.B>}, \vec{N} \in \mathscr{SN},\quad M_1^{\Gamma,W} \circ \vec{N} \in_{if} \mathscr{SN}\ \wedge\ M_2^{\Gamma,\pi x:A.B} \circ \vec{N} \in_{if} \mathscr{SN}$. Then one step reductions from $(M_1 \&^{W\cup\{\pi x:A.B\}} M_2) \circ \vec{N}$ will have only three possibilities: $M_1^{\Gamma,W} \circ \vec{N},\ M_2^{\Gamma,\pi x:A.B} \circ \vec{N}$ or $(M_1' \&^{W\cup\{\pi x:A.B\}} M_2') \circ \vec{N}'$, where in the last case just one of the primed terms is a one-step reduct of the corresponding non-primed one. The first two terms are possible only if they are well typed (subject-reduction) but in that case by assumption they are strongly normalizing. By induction on the maximal length of reduction of the tuple $< M_1, M_2, \vec{N} >$, we can prove that the last term is strongly normalizing. Therefore, we have $(M_1 \&^{W\cup\{\pi x:A.B\}} M_2) \circ \vec{N} \in_{if} \mathscr{SN}$. □

The $\beta_1^\&$ strong normalization follows.

THEOREM 5.24 ($\beta_1^\&$ Strong normalization). *In $\lambda\Pi^-_\&$, if $\Gamma \vdash N : A$, then N is $\beta_1^\&$ strongly normalizing.*

Proof. From The previous proposition and Corollary 5.22 □

6. PECULIAR PROPERTIES OF $\lambda\Pi_\leq$

$\lambda\Pi^\&$ is a conservative extension of $\lambda\Pi_\leq$ (see Section 4.6.1). Therefore, several properties we proved in the previous sections for $\lambda\Pi^\&$ hold for $\lambda\Pi_\leq$ as well. In particular $\lambda\Pi_\leq$ satisfies confluence, $\beta_2\Gamma$-strong normalization, admissibility of reflexivity and transitivity, and subject reduction.

However, some properties of $\lambda\Pi_\leq$ that are specific to it, and do not generalize to $\lambda\Pi^\&$. In this section, after having recalled the definition of $\lambda\Pi_\leq$, we study three of them:

1. We prove that the rule for subtyping family applications (rule S^{AC}-app in Section 3.3.2) is admissible in $\lambda\Pi_\leq$. We already explained in the excursion at the beginning of Section 4.4 why this proof is interesting for $\lambda\Pi_\leq$ but not for $\lambda\Pi^\&$. We think it is important to show this property here since it justifies the presence of the rule S-Λ in $\lambda\Pi_\leq$. [Section 6.2]

2. We show the equivalence between $\lambda\Pi_{\leq}$ and $\lambda P_{\leq}$ (see Section 3.3.2). This result is somewhat outside the main stream of this article. However, it is interesting to have it here since it shows the "roots" of $\lambda\Pi^{\&}$ and, more than the previous point, it justifies the definition of some rules of $\lambda\Pi^{\&}$ that have their form only to have this equivalence to hold. We simply outline the proof of equivalence; the full proof is available on the Web. [Section 6.3]

3. We prove the decidability of $\lambda\Pi_{\leq}$ and define a sound and complete algorithmic set of rules. This result is important since it forms the core of the study of decidability of $\lambda\Pi^{\&}$ of Section 7. Of course we could have studied decidability directly for $\lambda\Pi^{\&}$ without dealing with $\lambda\Pi_{\leq}$. We preferred to start by $\lambda\Pi_{\leq}$ for two reasons. First, it is interesting to show how the $\lambda\Pi_{\leq}$ algorithm modularly extends to $\lambda\Pi^{\&}$. Second, while decidability holds for $\lambda\Pi_{\leq}$, it does not hold for full $\lambda\Pi^{\&}$ but just for the normalizing subsystem studied in Section 5; so we preferred to show it also for a less powerful but full system such as $\lambda\Pi_{\leq}$, rather than just for a particular subsystem of $\lambda\Pi^{\&}$. [Section 6.4]

6.1. Definition of $\boldsymbol{\lambda}\Pi_{\leq}$

$\lambda\Pi_{\leq}$ has the same four syntactic categories as $\lambda\Pi^{\&}$, as well as the four judgment forms. The syntax of pre-terms, pre-types, pre-kinds and pre-contexts are as those of $\lambda\Pi^{\&}$ without overloaded types and terms.

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

There are only two notions of reduction, namely, $\rightarrow_{\beta_1}$ and $\rightarrow_{\beta_2}$. So, here, $\rightarrow_{\beta}\equiv\rightarrow_{\beta_1\cup\beta_2}$. The formation, kinding, and typing rules for $\lambda\Pi_{\leq}$ are those of $\lambda\Pi^{\&}$ (Appendix A.1) from which we erase K-OVER T-ϵ, T-$\&$, and T-OAPP. The subtyping rules are given in Figure 3. Note that, although S-ApR has the same form as in $\lambda\Pi^{\&}$, the β conversion occurring in there is only a combination of β_1 and β_2 ($\beta^{\&}$ is not involved).

6.2. Subtyping family application

The theorem of admissibility of the subtyping family application rule

$$\frac{\Gamma \vdash AM, BM : K \quad \Gamma \vdash A \leq B}{\Gamma \vdash AM \leq BM}$$

just requires a simple lemma:

LEMMA 6.1. $\Gamma \vdash \alpha M_1..M_n : K \ \wedge\ \alpha \textit{ bound in } \Gamma \ \Rightarrow\ \Gamma \vdash \Gamma(\alpha) M_1..M_n : K$

Proof. By the observation that there exist $A_1, .., A_n$ and K' such that $\alpha \leq \Gamma(\alpha) : \Pi x_1{:}A_1..\Pi x_n{:}A_n.K' \in \Gamma$ where $K'[\overline{x} := \overline{M}] = K$. □

THEOREM 6.1 (Subtyping family application).

$$\Gamma \vdash AM, BM : K \ \wedge\ \Gamma \vdash A \leq B \ \Rightarrow\ \Gamma \vdash AM \leq BM$$

Proof. By induction on the depth of derivation of $\Gamma \vdash A \leq B$. Cases (S-π) and (S-ApR) are immediate. For case (S-Λ) use Proposition 4.6 while for (S-ApT) use Lemma 6.1. In the case (S-ApSL) we have that the derivation ends by

$$\frac{\Gamma \vdash A''[x := M_1]M_2..M_n \leq B}{\Gamma \vdash (\Lambda x{:}A'.A'')M_1..M_n \leq B} \text{ S-ApSL}$$

From the assumption, we have:

$$\begin{array}{lll}
\Gamma \vdash ((\Lambda x{:}A'.A'')M_1..M_n)M : K & & \\
\quad \Rightarrow \ \exists D, K' \ s.t. \ \Gamma \vdash (\Lambda x{:}A'.A'')M_1..M_n : \Pi y{:}D.K' & & \textit{Prop. 4.14} \\
\qquad\qquad\qquad\qquad \wedge\ K = K'[x := M] \ \wedge \ \Gamma \vdash M : D & & \\
\quad \Rightarrow \ \Gamma \vdash A''[x := M_1]M_2..M_n : \Pi y{:}D.K' & & \textit{subject reduction} \\
\quad \Rightarrow \ \Gamma \vdash A''[x := M_1]M_2..M_n M : K'[y := M] & & \text{K-APP}
\end{array}$$

By the β_2 subject reduction (Lemma 4.21), $(A''[x := M_1])M_2..M_n M$ is well-kinded in the context Γ. Therefore,

$$\begin{array}{ll}
\Gamma \vdash A''[x := M_1]M_2..M_n M, BM : K \ \wedge \ \Gamma \vdash A''[x := M_1]M_2..M_n \leq B & \\
\quad \Rightarrow \ \Gamma \vdash A''[x := M_1]M_2..M_n M \leq BM & \textit{IH} \\
\quad \Rightarrow \ \Gamma \vdash (\Lambda x{:}A'.A'')M_1..M_n M \leq BM & \text{S-ApSL}
\end{array}$$

Case (S-ApSR) is similar to the last case. □

6.3. Equivalence between $\lambda\Pi_{\leq}$ and $\lambda P_{\leq}$

In this section we outline the proof of the equivalence between $\lambda\Pi_{\leq}$ and $\lambda P_{\leq}$. It can be skipped at first reading and requires the contents of Section 3.3.2.

Since the key difference between the two systems is in the definition of the subtyping relation, we concentrate our efforts on this relation. We expect the equivalence to state that when A and B have the same kind K, then $\Gamma \vdash A \leq B$ holds in our system if and only if $\Gamma \vdash^{AC} A \preceq B$ holds in $\lambda P_{\leq}$. But should A, B be kinded in $\lambda P_{\leq}$ or in $\lambda\Pi_{\leq}$? If the latter choice is taken, then the difficulty is to show that every subtyping rule in $\lambda\Pi_{\leq}$ is admissible in $\lambda P_{\leq}$ (since kindings may be different). If kindings are assumed in $\lambda P_{\leq}$, then the subject reduction in $\lambda\Pi_{\leq}$ does not apply and we do not know whether kinding is preserved (which is critical when β-reductions are involved in the proof).

So we take a different approach and prove equivalence by using a intermediate calculus $\lambda P^f_{\leq}$ that is defined by the same rules as $\lambda P_{\leq}$, except that the subtyping rules and the subsumption rule contain kinding judgments for all formulae occurring in them. We show that this system is equivalent both to $\lambda\Pi_{\leq}$ and $\lambda P_{\leq}$. We use these two equivalence results to prove the equivalence between $\lambda\Pi_{\leq}$ and $\lambda P_{\leq}$ and in particular to give an answer to the question of the previous paragraph, showing that the equivalence of the subtyping relations must be stated by using $\lambda\Pi_{\leq}$ kinding.

Once more in order to avoid confusion we use a different relation symbol $\preccurlyeq$ to denote $\lambda P^f_{\leq}$ subtyping, use f scripts for $\lambda P^f_{\leq}$ rules and judgments, and use lowercase italicized names for rules. The subtyping rules of $\lambda P^f_{\leq}$ are:

$$S^f\textit{-var}\quad \frac{\alpha \textit{ bounded in } \Gamma}{\Gamma \vdash_f \alpha \preccurlyeq \Gamma(\alpha)}$$

$$S^f\text{-}\pi\quad \frac{\Gamma \vdash_f \pi x{:}A.B, \pi x{:}A'.B' : \star \quad \Gamma \vdash_f A' \preccurlyeq A,\ \ \Gamma, x : A' \vdash_f B \preccurlyeq B'}{\Gamma \vdash_f \pi x{:}A.B \preccurlyeq \pi x{:}A'.B'}$$

$$S^f\text{-}\Lambda\quad \frac{\Gamma \vdash_f \Lambda x{:}A.B, \Lambda x{:}A'.B' : K \quad A' =_\beta A \quad \Gamma, x : A \vdash B \preccurlyeq B'}{\Gamma \vdash_f \Lambda x{:}A.B \preccurlyeq \Lambda x{:}A.B'}$$

$$S^f\textit{-app}\quad \frac{\Gamma \vdash_f AM, BM : K \quad \Gamma \vdash_f A \preccurlyeq B}{\Gamma \vdash_f AM \preccurlyeq BM}$$

$$S^f\textit{-conv}\quad \frac{\Gamma \vdash_f A, B : K \quad A =_\beta B}{\Gamma \vdash_f A \preccurlyeq B}$$

$$S^f\textit{-trans}\quad \frac{\Gamma \vdash_f A, B, C : K \quad \Gamma \vdash_f A \preccurlyeq B \quad \Gamma \vdash_f B \preccurlyeq C}{\Gamma \vdash_f A \preccurlyeq C}$$

$\lambda P^f_{\leq}$ differs from $\lambda P_{\leq}$ also in the subsumption rule which contains kinding judgments for the types at issue (as in $\lambda\Pi^{\&}$):

$$T^f\textit{-sub}\ \frac{\Gamma \vdash_f M : A \quad \Gamma \vdash_f A \preccurlyeq B \quad \Gamma \vdash_f A, B : \star}{\Gamma \vdash_f M : B}$$

As announced we do not give a detailed proof of equivalence, but we rather outline it. The interested reader will find full proofs in [Che96, Che98] available on the Web.

First, it is easy to verify that some structural properties proved in Section 4.3 for $\lambda\Pi^{\&}$ hold for $\lambda P^f_{\leq}$, as well: generation for kinding, context properties, uniqueness of kinds, bound β-equivalence, and agreement of judgments. However, the proofs differ from those of $\lambda\Pi^{\&}$ in that $\lambda P^f_{\leq}$ requires simultaneous induction on formation, kinding, typing, and subtyping, while in $\lambda\Pi^{\&}$ (or $\lambda\Pi_{\leq}$) two separated inductions can be used, one for the first three judgments, another for subtyping.

The circularity between kinding, typing, and subtyping in $\lambda P^f_{\leq}$ is also the central difficulty in proving its equivalence with $\lambda\Pi_{\leq}$. We handle it by proving the results in the following order (where J denotes either a kind K, or a kinding $A{:}K$, or a typing $M{:}A$, but not a subtyping relation):

1. $\Gamma \vdash_f A, B : K \ \wedge\ A =_\beta B \ \Rightarrow\ \Gamma \vdash A \leq B$
2. $\Gamma \vdash_f AM, BM : K \ \wedge\ \Gamma \vdash A \leq B \ \Rightarrow\ \Gamma \vdash AM \leq BM$
3. $\Gamma \vdash_f A \preccurlyeq B \Rightarrow \Gamma \vdash A \leq B$
4. $\Gamma \vdash A \leq B \ \wedge\ \Gamma \vdash A, B : K \ \Rightarrow\ \Gamma \vdash_f A \preccurlyeq B$
5. $\Gamma \vdash J \ \Rightarrow\ \Gamma \vdash_f J$
6. $\Gamma \vdash_f J \ \Rightarrow\ \Gamma \vdash J$
7. $(\Gamma \vdash_f A \preccurlyeq B \ \Rightarrow \Gamma \vdash_f A, B : K) \ \wedge\ (\Gamma \vdash_f A, B : K \Rightarrow\ \Gamma \vdash A, B : K)$
8. $\Gamma \vdash_f A \preccurlyeq B \ \Leftrightarrow\ \Gamma \vdash A \leq B \ \wedge\ \Gamma \vdash A, B : K$

We can now precisely state the equivalence between $\lambda P^f_{\leq}$ and $\lambda\Pi_{\leq}$.

THEOREM 6.2 (Equivalence between $\lambda P^f_{\leq}$ and $\lambda\Pi_{\leq}$).

$$\begin{array}{lcl} \Gamma \vdash_f K & \Leftrightarrow & \Gamma \vdash K \\ \Gamma \vdash_f A : K & \Leftrightarrow & \Gamma \vdash A : K \\ \Gamma \vdash_f M : A & \Leftrightarrow & \Gamma \vdash M : A \\ \Gamma \vdash_f A \preccurlyeq B & \Leftrightarrow & \Gamma \vdash A \leq B \ \wedge \ \Gamma \vdash A : K \ \wedge \ \Gamma \vdash B : K \end{array}$$

It still remains to prove the equivalence between $\lambda P^f_{\leq}$ and $\lambda P_{\leq}$. The proof is quite straightforward. (Recall that $\preceq$ and $\vdash^{AC}$ respectively denote the subtyping relation and judgments (derivable) in Aspinall and Compagnoni's system $\lambda P_{\leq}$).

THEOREM 6.2 (Equivalence between $\lambda P^f_{\leq}$ and $\lambda P_{\leq}$).

$$\begin{array}{lcl} \Gamma \vdash_f K & \Leftrightarrow & \Gamma \vdash^{AC} K \\ \Gamma \vdash_f A : K & \Leftrightarrow & \Gamma \vdash^{AC} A : K \\ \Gamma \vdash_f M : A & \Leftrightarrow & \Gamma \vdash^{AC} M : A \\ \Gamma \vdash_f A \preccurlyeq B & \Leftrightarrow & \Gamma \vdash^{AC} A \preceq B \end{array}$$

Proof. By simultaneous induction, by using the agreement of judgments for $\lambda P^f_{\leq}$. $\square$

The equivalence between $\lambda\Pi_{\leq}$ and $\lambda P_{\leq}$ then follows:

COROLLARY 6.1 (Equivalence between $\lambda P_{\leq}$ and $\lambda\Pi_{\leq}$).

$$\begin{array}{lcl} \Gamma \vdash^{AC} K & \Leftrightarrow & \Gamma \vdash K \\ \Gamma \vdash^{AC} A : K & \Leftrightarrow & \Gamma \vdash A : K \\ \Gamma \vdash^{AC} M : A & \Leftrightarrow & \Gamma \vdash M : A \\ \Gamma \vdash^{AC} A \preceq B & \Leftrightarrow & \Gamma \vdash A \leq B \ \wedge \ \Gamma \vdash A : K \ \wedge \ \Gamma \vdash B : K \end{array}$$

EXCURSUS ON RECENT WORK *We already explained that $\lambda\Pi_{\leq}$ is a byproduct of $\lambda\Pi^{\&}$ since it was defined in preparation to this work. However, $\lambda\Pi_{\leq}$ is not deprived of interest on its own. Although we just proved that $\lambda\Pi_{\leq}$ is equivalent to $\lambda P_{\leq}$, in Section 3.3.2 we argued that $\lambda\Pi_{\leq}$ improves $\lambda P_{\leq}$ in that it allows type level transitivity elimination in subtyping. This is obtained thanks to the rules S-ApSR and S-ApSL that embed β_2-head-reduction in the subtyping rules. These rules have been generalized in [Che97] (a study on the extension of the Calculus of Constructions by subtyping), to S-β:*

$$\frac{A =_\beta C \quad \Gamma \vdash C \leq D \quad D =_\beta B \quad \Gamma \vdash A, B, C, D : s}{\Gamma \vdash A \leq B}$$

where $\Gamma \vdash A, B, C, D : s$ denotes that A, B, C, D are well-formed, and to S-β' (for the algorithmic subtyping system)[28] *:*

$$\frac{A \rightarrow_\beta C \quad \Gamma \vdash C \leq D \quad B \rightarrow_\beta D}{\Gamma \vdash A \leq B}$$

Several authors have used similar techniques. In their work on typed operational semantics for (a variant of) $F^\omega_{\leq}$, Compagnoni and Goguen [CG97] use the subtyping rule

$$\frac{A \rightarrow_w C \quad \Gamma \vdash C \leq D \quad B \rightarrow_w D}{\Gamma \vdash A \leq B}$$

[28] A detailed analysis on this technique of achieving transitivity elimination can be found in [Che98].

where $\rightarrow_w$ denotes weak-head reduction.

In his study on coercive subtyping for UTT (a system more expressive than the Calculus of Constructions) Luo [Luo97] deals with type conversion by the following rule:

$$\frac{A = C \quad \Gamma \vdash C \leq_{c'} D \quad D = B \quad \Gamma \vdash A, B, C, D : \star \quad c = c'}{\Gamma \vdash A \leq_c B}$$

where c, c' are coercions, $=$ is the type conversion defined in his system and $\Gamma \vdash A, B, C, D : \star$ states that A, B, C, D are well-formed.

A significant achievement in this direction is the work of Zwanenburg [Zwa99], where transitivity elimination for general PTS systems is obtained by the rule

$$\frac{A \rightarrow_\beta C \quad \Gamma \vdash C \leq D \quad B \rightarrow_\beta D}{\Gamma \vdash A \leq B}$$

Although a very similar to the S-β' rule in [Che97], Zwanenburg is the first who provides a direct proof of transitivity elimination with this rule. This progress allows him to construct and study the subtyping extension to general PTS systems, which includes the one of [Che97].

The common feature of all these approaches is that the resulting subtyping systems enjoy the transitivity elimination property. As in this work, the solution of this problem is the key step in their studies of meta theoretic properties.

Compared to these recent and more general approaches the pair of rules S-ApSL and S-ApSR is still interesting: it is simple (as part of a subtyping system) and efficient (as part of a subtype checking algorithm). $\square$

6.4. Decidability and Minimal Typing

We already hinted that the set of subtyping rules for $\lambda\Pi_{\leq}$ can be straightforwardly turned into a deterministic algorithm by adding to the [S-ApSR] rule the condition $C \not\equiv (\Lambda x{:}A'.B')M_1'..M_m' \wedge C \not\equiv \alpha M_1' \dots M_m'$ and to the rule [S-ApT] the condition $A \not\equiv \alpha M_1' \dots M_n'$ (Footnote 9 in Section 3.3.2).

Of course, it is necessary to prove that every judgment provable by the unrestricted rules can also be proved just by using the rules with the extra conditions (the converse is straightforward), that is, we have to show that the conditions are neutral with respect to the definition of the subtyping relation.

Let us briefly hint to how this equivalence can be proved:

Proof. To show that the condition on the rule [S-ApT] is neutral it suffices to see that whenever $\Gamma \vdash \alpha M_1 \dots M_n \leq \alpha N_1 \dots N_n$ then $M_i =_\beta N_i$ (observe the rules and note that the only way to decompose a variable on the right-hand side is to use the rule [S-ApR]). Therefore if we have a proof of $\Gamma \vdash \alpha M_1 \dots M_n \leq \alpha N_1 \dots N_n$ ending by [S-ApT] then we can prove the same judgment just by using [S-ApR].

The proof that the conditions on [S-ApSR] are neutrals is instead obtained by induction on the depth of the derivations (with the extra result that the derivation that satisfies the condition does not have greater depth). For example consider the case in which a derivation of $\Gamma \vdash (\Lambda y{:}C.D)M_1 \dots M_m \leq (\Lambda x{:}A.B)N_1 \dots N_n$ ends by the rule [S-ApSR]; then by induction hypothesis there exists a derivation for $\Gamma \vdash (\Lambda y{:}C.D)M_1 \dots M_m \leq B[x :=$

$N_1]N_2 \ldots N_n$ that satisfies the conditions. Therefore, the last rule of this derivation must be [S-ApSL], whence $\Gamma \vdash D[y := M_1]M_2 \ldots M_m \leq B[x := N_1]N_2 \ldots N_n$. By applying [S-ApSR] we deduce $\Gamma \vdash D[y := M_1]M_2 \ldots M_m \leq (\Lambda x{:}\, A.B)N_1 \ldots N_n$. This last rule either satisfies the conditions or it does not. In the first case we already have a whole derivation that satisfies the conditions; in the latter case we can apply once more the induction hypothesis and obtain such a derivation. In both cases a further application of [S-ApSL] rule yields a derivation of the initial judgment that satisfies the conditions. ☐

The decidability of subtyping follows from the equivalence between $\lambda\Pi_{\leq}$ and $\lambda P_{\leq}$ and the decidability of the latter, but it can also be easily obtained directly from the subtyping rules of $\lambda\Pi_{\leq}$.

THEOREM 6.3 (Decidability of subtyping). *If* $\Gamma \vdash A{:}\, K_1, \Gamma \vdash B{:}\, K_2$*, then the subtyping judgment* $\Gamma \vdash A \leq B$ *is decidable.*

Proof. Associate to each subtyping judgment $\Gamma \vdash A \leq B$ the measure *Weight*$_\Gamma(A, B)$ of Section 4.4.1 and note that each subtyping rule decreases this measure. ☐

The next step towards proving decidability is to design algorithmic versions of the remaining judgments (typing, kinding, and context formation). Here we describe only the most significative algorithmic rules: all the algorithmic rules can be found in Appendix A.1.3 (just remove the rules specific to $\lambda\Pi^{\&}$, that is, AS-OVER, Lub-OVER, AK-OVER, AT-ε, AT-&, and AT-OAPP).

Judgments in algorithmic rules are denoted by $\Gamma \vdash_{\mathscr{A}} J$. In particular, we write $\Gamma \vdash_{\mathscr{A}} A \leq B$ to denote judgments deduced by using the rules with the extra conditions at the very beginning of this section (even though in what follows we tend to omit the $\mathscr{A}$ script for subtyping since they virtually denote the same system). With the convention that premises are evaluated in order, the rules form an algorithm.

The main step to an algorithmic set of rules is as customary: we remove the subsumption rule (which is not syntax-directed) and embed the subtyping relation in the elimination rules (those for applications, namely K-APP and T-APP). As usual, the presence of type variables causes the further problem that the type to eliminate in a elimination rule may not be in "canonical" form. For example, consider the application of two terms MN under a context Γ. We first try to type each term $\Gamma \vdash M{:}\, A$ and $\Gamma \vdash N{:}\, B$, and then to infer from that a type C such that $\Gamma \vdash MN : C$. When the type A of M is equivalent to a type of the form $\pi x{:}D.E$, then it suffices to check that $\Gamma \vdash B \leq D$ and to infer that C is $E[x := N]$. But because of type variables the actual form of A may be $\alpha M_1..M_n$ or $(\Lambda y{:}A_1.A_2)M_1..M_n$. Therefore, we need to infer from these types a type of the form $\pi x{:}D.E$ (let us call it a π-type). In $\lambda P_{\leq}$, this is achieved by using a function $\text{FLUB}_\Gamma(A)$ that returns the least π-type super-type of A (strictly speaking, it returns *a* minimal π-type super-type of A). Here we essentially take the same approach with the only difference that the least π-type is inferred rather than calculated. So we introduce a new relation

$$\Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} B$$

to express the fact that B is the least π-type super-type of A under the context Γ. The rules to derive this relation are:

$$\text{Lub-REFL}\quad \frac{}{\Gamma \vdash_{\mathscr{A}} \pi x{:}A.B \leq_{\pi lub} \pi x{:}A.B}$$

$$\text{Lub-ApT}\quad \frac{\Gamma \vdash_{\mathscr{A}} \Gamma(\alpha)M_1..M_n \leq_{\pi lub} A}{\Gamma \vdash_{\mathscr{A}} \alpha M_1..M_n \leq_{\pi lub} A}$$

$$\text{Lub-ApSL}\quad \frac{\Gamma \vdash_{\mathscr{A}} B[x := M_1]M_2..M_n \leq_{\pi lub} C}{\Gamma \vdash_{\mathscr{A}} (\Lambda x{:}A.B)M_1..M_n \leq_{\pi lub} C}$$

The properties of this relation are stated by the following proposition.

PROPOSITION 6.3 (Properties of πlub judgments).

1.$\Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} B \Rightarrow \Gamma \vdash A \leq B$

2.$\Gamma \vdash A \leq \pi x{:}B.C \Rightarrow \exists B', C'\ s.t.\ \Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} \pi x{:}B'.C' \wedge \Gamma \vdash \pi x{:}B'.C' \leq \pi x{:}B.C$

3.Given a type A, it is decidable if there exists $\pi x{:}B.C$ such that $\Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} \pi x{:}B.C$ is derivable.

4.$\Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} B \wedge \Gamma \vdash A : K \Rightarrow \Gamma \vdash B : K$

5.$\Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} \pi x{:}B.C \wedge \Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} \pi x{:}B'.C' \Rightarrow B \equiv B' \wedge C \equiv C'$

Proof. The fourth claim is proved by induction on the depth of the derivation of $\Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} B$ using subject reduction. The others are straightforward. □

While the use of a FLUB function or of the πlub judgment is a matter of style, the key difference between our approach and $\lambda P_{\leq}$ is in the typing and kinding application rules. We define subtyping directly on types without β_2 normalizing them:

$$\text{AK-APP}\quad \frac{\Gamma \vdash_{\mathscr{A}} A : \Pi x{:}B.K \quad \Gamma \vdash_{\mathscr{A}} M : B' \quad \Gamma \vdash_{\mathscr{A}} B' \leq B}{\Gamma \vdash_{\mathscr{A}} AM : K[x := M]}$$

$$\text{AT-APP}\quad \frac{\Gamma \vdash_{\mathscr{A}} M : A \quad \Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} \pi x : B.C \quad \Gamma \vdash_{\mathscr{A}} N : B' \quad \Gamma \vdash_{\mathscr{A}} B' \leq B}{\Gamma \vdash_{\mathscr{A}} MN : C[x := N]}$$

In the same way as we removed the subsumption rule by embedding subtyping in the applications rules, we eliminate the conversion rule K-CONV by embedding conversions in the context-formation rule F-SUBTYPE.

To obtain an algorithmic system we also remove from T-VAR and K-VAR the context-formation premises (yielding AT-VAR and AK-VAR of Appendix A.1.3). In this way typing and kinding become independent from context formation. Therefore, $\Gamma \vdash_{\mathscr{A}} J$ no longer implies $\Gamma \vdash \star$. So additional kinding checks $\Gamma \vdash_{\mathscr{A}} A : \star$ must be added to the introduction rules (rules AK-π, AK-Λ, and AT-λ) and an additional kinding check $\Gamma \vdash_{\mathscr{A}} K$ must be added to context formation rule F-SUBTYPE. This suffices to infer the well-kindedness of contexts in the remaining rules.

Proofs of soundness and completeness of the algorithmic system are a little longer than those in $\lambda P_{\leq}$, but do not require any specific technique or insight.

For the proof of soundness of the algorithmic rules, we need to ensure that well-kindedness of types in the premises of subsumption is satisfied. This is achieved by using the properties of πlub judgments and generation for kinding.

THEOREM 6.4 (Soundness of algorithmic system). *For all* Γ, A, K, M*,*

$$\begin{array}{lll} 1. & \Gamma \vdash_{\mathscr{A}} K & \Rightarrow \Gamma \vdash K \\ 2. & \Gamma \vdash \star \wedge \Gamma \vdash_{\mathscr{A}} A : K & \Rightarrow \Gamma \vdash A : K \\ 3. & \Gamma \vdash \star \wedge \Gamma \vdash_{\mathscr{A}} M : A & \Rightarrow \Gamma \vdash M : A \end{array}$$

Proof. Simultaneously by induction on the depth of the derivation in the algorithmic system.

CASE (AT-APP). Suppose we have a derivation ended by an application of the rule AT-APP

$$\frac{\Gamma \vdash_{\mathscr{A}} M : A \quad \Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} \pi x : B.C \quad \Gamma \vdash_{\mathscr{A}} N : B' \quad \Gamma \vdash_{\mathscr{A}} B' \leq B}{\Gamma \vdash_{\mathscr{A}} MN : C[x := N]} \text{ AT-APP}$$

We have:

$$\begin{array}{llll} \Gamma \vdash \star \wedge \Gamma \vdash_{\mathscr{A}} M : A & \Rightarrow & \Gamma \vdash M : A & \textit{IH} \\ & \Rightarrow & \Gamma \vdash A : \star & \textit{Proposition 4.13} \\ \Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} \pi x{:}B.C & \Rightarrow & \Gamma \vdash A \leq \pi x{:}B.C & \textit{Proposition 6.3(1)} \\ \Gamma \vdash \star \wedge \Gamma \vdash_{\mathscr{A}} N : B' & \Rightarrow & \Gamma \vdash N : B' & \textit{IH} \\ & \Rightarrow & \Gamma \vdash B' : \star & \textit{Proposition 4.13} \\ \Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} \pi x{:}B.C \wedge \Gamma \vdash A : \star & \Rightarrow & \Gamma \vdash \pi x{:}B.C : \star & \textit{Proposition 6.3(4)} \\ & \Rightarrow & \Gamma \vdash B : \star & \textit{Propositions 4.14, 4.10} \end{array}$$

So we have the following derivation ending by an instance of the rule T-APP, and whose premises are derived by two instances of the rule T-SUB:

$$\frac{\dfrac{\Gamma \vdash M{:}A \quad \Gamma \vdash A \leq \pi x{:}B.C \quad \Gamma \vdash A, \pi x{:}B.C : \star}{\Gamma \vdash M : \pi x{:}B.C} \quad \dfrac{\Gamma \vdash N{:}B' \quad \Gamma \vdash B' \leq B \quad \Gamma \vdash B', B : \star}{\Gamma \vdash N : B}}{\Gamma \vdash MN : C[x := N]}$$

The case for AK-APP is similar. Others are easy. □

COROLLARY 6.4.

$$\begin{array}{l} \Gamma \vdash_{\mathscr{A}} \star \wedge \Gamma \vdash_{\mathscr{A}} A : K \Rightarrow \Gamma \vdash A : K \\ \Gamma \vdash_{\mathscr{A}} \star \wedge \Gamma \vdash_{\mathscr{A}} M : A \Rightarrow \Gamma \vdash M : A \end{array}$$

As usual in the presence of subtyping and/or typing conversion the algorithmic system does not prove all the judgments of the original system. Nevertheless it is complete in the sense that every context, type, or term that is well-kinded/typed in the original system is so in the algorithmic one:

THEOREM 6.5 (Completeness of algorithmic system).

$$\begin{array}{lll} 1. & \Gamma \vdash K & \Rightarrow \Gamma \vdash_{\mathscr{A}} K \\ 2. & \Gamma \vdash A : K & \Rightarrow \exists K_a \; s.t. \; \Gamma \vdash_{\mathscr{A}} A : K_a \wedge K_a =_\beta K \wedge \Gamma \vdash K_a \\ 3. & \Gamma \vdash M : A & \Rightarrow \exists A_a \; s.t. \; \Gamma \vdash_{\mathscr{A}} M : A_a \wedge \Gamma \vdash A_a \leq A \end{array}$$

Proof. Simultaneously by induction on derivations in the original system.
<u>CASE</u> (K-APP). Suppose the last step of derivation is

$$\frac{\Gamma \vdash A : \Pi x{:}B.K \quad \Gamma \vdash M : B}{\Gamma \vdash AM : K[x := M]} \; K\text{-APP}$$

We have

$$\begin{array}{lll}
\Gamma \vdash A : \Pi x{:}B.K & \Rightarrow\ \exists K_a \; s.t. \; \Gamma \vdash_{\mathscr{A}} A : K_a \;\wedge\; K_a =_\beta \Pi x{:}B.K & \textit{IH} \\
& \Rightarrow\ \exists B', K' \; s.t. \; K_a \equiv \Pi x{:}B'.K' \;\wedge & \\
& \quad\ B' =_\beta B \;\wedge\; K' =_\beta K & \textit{confluence} \\
\Gamma \vdash M : B & \Rightarrow\ \Gamma \vdash_{\mathscr{A}} M : B'' \;\wedge\; \Gamma \vdash B'' \leq B & \textit{IH} \\
\Gamma \vdash M : B & \Rightarrow\ \Gamma \vdash \star \;\wedge\; \Gamma \vdash B : \star & \textit{Prop. 4.10, 4.13} \\
\Gamma \vdash \star \;\wedge\; \Gamma \vdash_{\mathscr{A}} A : K_a & \Rightarrow\ \Gamma \vdash A : K_a & \textit{Theor. 6.4} \\
& \Rightarrow\ \Gamma \vdash K_a & \textit{Prop. 4.13} \\
& \Rightarrow\ \Gamma \vdash B' : \star & \text{F-}\Pi\text{, } \textit{Prop. 4.13} \\
\Gamma \vdash B, B' : \star \;\wedge\; B' =_\beta B & \Rightarrow\ \Gamma \vdash B \leq B' & \textit{Prop. 4.24} \\
& \Rightarrow\ \Gamma \vdash B'' \leq B' & \textit{Prop. 4.27} \\
& \Rightarrow\ \Gamma \vdash_{\mathscr{A}} B'' \leq B' &
\end{array}$$

So we have a derivation ending by

$$\frac{\Gamma \vdash_{\mathscr{A}} A : \Pi x{:}B'.K' \quad \Gamma \vdash_{\mathscr{A}} M : B'' \quad \Gamma \vdash_{\mathscr{A}} B'' \leq B'}{\Gamma \vdash_{\mathscr{A}} AM : K'[x := M]} \; \text{AK-APP}$$

and $K'[x := M] =_\beta K[x := M]$.

<u>CASE</u> (T-APP). Suppose the derivation ends by

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

Then there exist C, A', B', A'' such that

$$\begin{array}{lll}
\Gamma \vdash M : \pi x{:}A.B & \Rightarrow\ \Gamma \vdash_{\mathscr{A}} M : C \;\wedge\; \Gamma \vdash C \leq \pi x{:}A.B & \textit{IH} \\
\Gamma \vdash C \leq \pi x{:}A.B & \Rightarrow\ \Gamma \vdash_{\mathscr{A}} C \leq_{\pi lub} \pi x{:}A'.B' \wedge \Gamma \vdash \pi x{:}A'.B' \leq \pi x{:}A.B & \textit{Prop. 6.3} \\
& \Rightarrow\ \Gamma \vdash A \leq A' & \textit{Prop. 4.14} \\
\Gamma \vdash N : A & \Rightarrow\ \Gamma \vdash_{\mathscr{A}} N : A'' \;\wedge\; \Gamma \vdash A'' \leq A & \textit{IH} \\
& \Rightarrow\ \Gamma \vdash A'' \leq A' & \textit{Prop. 4.27} \\
& \Rightarrow\ \Gamma \vdash_{\mathscr{A}} A'' \leq A' &
\end{array}$$

Therefore, we have a derivation ending by

$$\frac{\Gamma \vdash_{\mathscr{A}} M : C \quad \Gamma \vdash_{\mathscr{A}} C \leq_{\pi lub} \pi x{:}A'.B' \quad \Gamma \vdash_{\mathscr{A}} N : A'' \quad \Gamma \vdash_{\mathscr{A}} A'' \leq A'}{\Gamma \vdash_{\mathscr{A}} MN : B'[x := N]} \; \text{AT-APP}$$

and $\Gamma \vdash B'[x := N] \leq B[x := N]$ follows from Proposition 4.11.

Other cases are easy. $\square$

In the proof of completeness, we have used reflexivity and transitivity of subtyping. To apply reflexivity, we need kinding condition, which is proved by context properties, agreement of judgments, and soundness of the algorithmic rules. Generation for kinding has been used to decompose subtyping between π-types so that transitivity can apply.

A *minimal type* of a term M under a context Γ is a type A such that $\Gamma \vdash M : A$ and for any other type B if $\Gamma \vdash M : B$, then $\Gamma \vdash A \leq B$. Note that, by this definition, the minimal type of a term M may not be unique: if B is a minimal type of M, then every well-formed type β-equivalent to B is also a minimal type of M. But our algorithmic system will always return the same minimal type. This type may not be in normal form. For example, if $x : A \in \Gamma$, then the algorithm returns A for x, even if A is not in normal form. To obtain the minimal normalized type of a term, one can simply normalize the type returned by the algorithm.

In order to show the minimal typing property, it remains to show that the A_a in the proposition of completeness is unique.

PROPOSITION 6.5 (Uniqueness and minimality of algorithmic typing).

$$\Gamma \vdash_{\mathscr{A}} M : A \ \wedge\ \Gamma \vdash_{\mathscr{A}} M : B \ \Rightarrow\ A \equiv B$$
$$\Gamma \vdash_{\mathscr{A}} M : A \ \wedge\ \Gamma \vdash M : B \ \Rightarrow\ \Gamma \vdash A \leq B$$

Proof. The first implication is proved by induction on the size of M, using uniqueness of πlub (Proposition 6.3). The second implication follows from Theorem 6.5 . □

The minimal typing property follows.

COROLLARY 6.6 (Minimal typing property for $\lambda\Pi_{\leq}$).

$$\Gamma \vdash M : A \ \Rightarrow\ (\exists B\ s.t.\ \Gamma \vdash M : B \ \wedge\ \forall C\ \ \Gamma \vdash M : C \Rightarrow \Gamma \vdash B \leq C)$$

Decidability results can be straightforwardly proved by showing that the algorithms always terminate. They are summarized in the following proposition.

PROPOSITION 6.7 (Decidability of algorithmic $\lambda\Pi_{\leq}$). *For all* Γ, K, M, A *and* B, *the following problems are decidable*[29] *:*

1.$\Gamma \vdash A : K \ \wedge\ \Gamma \vdash B : K' \ \Rightarrow\ \Gamma \vdash_{\mathscr{A}} A \leq B$ *?*
2.$\Gamma \vdash \star \ \Rightarrow\ \exists K\ s.t.\ \Gamma \vdash_{\mathscr{A}} A : K$ *?*
3.$\Gamma \vdash \star \ \Rightarrow\ \exists A\ s.t.\ \Gamma \vdash_{\mathscr{A}} M : A$ *?*
4.$\Gamma \vdash_{\mathscr{A}} K$ *?*

Proof. The assertions must be proved in the order shown. The first implication was proved in Theorem 6.3. The second and third implications are proved simultaneously by setting *Weight*$(\Gamma \vdash_{\mathscr{A}} U{:}V) =$ *Size*(U) and showing that each algorithmic typing and kinding rule strictly decreases *Weight*. For the last implication set *Size*$(<>) = 1$, *Size*$(\Gamma, \alpha{:}K) =$ *Size*$(\Gamma, \alpha \leq A{:}K) =$ *Size*$(\Gamma) +$ *Size*(K), and *Size*$(\Gamma, x{:}A) =$ *Size*(Γ); then note that all the algorithmic formation rules strictly decrease *Weight*$(\Gamma \vdash_{\mathscr{A}} K)$ defined as the lexicographical order of the following pair: $($*Size*$(\Gamma) +$ *Size*(K), *length*$(\Gamma))$. □

[29] We say that the problem $\phi \Rightarrow \psi\Gamma$ is decidable if under the hypothesis ϕ, the formula ψ is decidable. As usual $\Gamma \vdash J$ stands for "$\Gamma \vdash J$ *is derivable*".

By the equivalence between algorithmic rules and the original ones, we obtain the decidability of judgments in $\lambda\Pi_{\leq}$.

COROLLARY 6.8 (Decidability of $\lambda\Pi_{\leq}$). *The following problems are decidable: for all* Γ, K, M, A *and* B,

1.$\Gamma \vdash A : K \;\wedge\; \Gamma \vdash B : K' \;\Rightarrow\; \Gamma \vdash A \leq B$ *?*
2.$\Gamma \vdash \star \;\Rightarrow\; \exists K\ s.t.\ \Gamma \vdash A : K$ *?*
3.$\Gamma \vdash \star \;\Rightarrow\; \exists A\ s.t.\ \Gamma \vdash M : A$ *?*
4.$\Gamma \vdash K$ *?*

7. DECIDABILITY AND ALGORITHMIC SYSTEM FOR $\lambda\Pi^{\&}$

In this section we show how the proofs of Section 6.4 for $\lambda\Pi_{\leq}$ can be lifted to $\lambda\Pi^{\&}$. In particular we examine the algorithmic type system for $\lambda\Pi^{\&}$, its soundness, its completeness, and we discuss the minimal type property and decidability results.

The algorithmic subtyping system for $\lambda\Pi^{\&}$ is obtained by adding the subtyping rule for overloaded types S-OVER to the algorithmic subtyping system of $\lambda\Pi_{\leq}$ (that is, by adding suitable conditions to the subtyping rules of $\lambda\Pi^{\&}$ as explained at the beginning of Section 6.4) . The algorithmic subtyping system is equivalent to the original one:

$$\Gamma \vdash S \leq T \;\Leftrightarrow \Gamma \vdash_{\mathscr{A}} S \leq T$$

where $\vdash_{\mathscr{A}}$ denotes judgments of the algorithmic system. The proof of this equivalence is strictly the same as the one outlined in the previous section for $\lambda\Pi_{\leq}$.

For any subsystem of $\lambda\Pi^{\&}$ in which the corresponding β-equivalence is decidable, the termination of the algorithm can be proved in a similar way as $\lambda\Pi_{\leq}$, that is by using the measure *Weight*. In other words, the subtyping is decidable in every subsystem of $\lambda\Pi^{\&}$ with decidable β-equivalence. An example of such a subsystem is the system $\lambda\Pi^{-}_{\&}$ we introduced in Section 5.

The whole algorithmic system is obtained by a few modifications to the algorithmic system of $\lambda\Pi_{\leq}$ and it is summarized in Appendix A.1.3. First, the set of πlub rules is extended by a new rule

Lub-OVER $$\frac{}{\Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n} \leq_{\pi lub} \{\pi x{:}A_i.B_i\}_{i\leq n}}$$

Recall that, in $\lambda\Pi_{\leq}$, the judgment $\Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} B$ is used to infer the least π-type super-type of A. With the new rule, it will infer either the least π-type or the least overloaded-type super-type of A.

It is easy to verify that all properties for the relation $\Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} B$ (Proposition 6.3) still hold. In addition, we have a new property:

$$\begin{aligned}&\Gamma \vdash C \leq \{\pi x{:}A_i.B_i\}_{i\leq n}\\ &\qquad\Rightarrow\; \Gamma \vdash_{\mathscr{A}} C \leq_{\pi lub} \{\pi x{:}A'_j.B'_j\}_{j\leq m} \;\wedge\; \Gamma \vdash \{\pi x{:}A'_j.B'_j\}_{j\leq m} \leq \{\pi x{:}A_i.B_i\}_{i\leq n}\end{aligned}$$

The algorithmic context rules are the same as those for $\lambda\Pi_{\leq}$. We add K-OVER to algorithmic kinding rules of $\lambda\Pi_{\leq}$, while the following rules are added to the algorithmic typing rules of $\lambda\Pi_{\leq}$:

AT-ε
$$\frac{\Gamma \vdash_{\mathscr{A}} \star}{\Gamma \vdash_{\mathscr{A}} \varepsilon : \{\}}$$

AT-&
$$\frac{\begin{array}{cc}\Gamma \vdash_{\mathscr{A}} M : W_1 \leq \{\pi x{:}A_i.B_i\}_{i\leq n} & \Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n} : \star \\ \Gamma \vdash_{\mathscr{A}} N : W_2 \leq \pi x{:}A_{n+1}.B_{n+1} & \Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star\end{array}}{\Gamma \vdash_{\mathscr{A}} M \&^{\{\pi x:A_i.B_i\}_{i\leq n+1}} N : \{\pi x{:}A_i.B_i\}_{i\leq n+1}}$$

AT-OAPP
$$\frac{\begin{array}{cc}\Gamma \vdash_{\mathscr{A}} M : W \leq_{\pi lub} \{\pi x{:}A_i.B_i\}_{i\leq n+1} & \Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star \\ \Gamma \vdash_{\mathscr{A}} N : A & A_j = \min_{i\leq n+1}\{A_i \mid \Gamma \vdash A \leq A_i\}\end{array}}{\Gamma \vdash_{\mathscr{A}} M \bullet N : B_j[x := N]}$$

Note that in the rule AT-& the hypothesis $\Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star$ does not imply $\Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n} : \star$, so both kinding hypothesis are needed.

As in the case of $\lambda\Pi_{\leq}$, using induction and the subsumption rule, it is easy to prove the soundness of the algorithmic system.

NOTATION 7.1. *We use* $\Gamma \vdash M : A \leq B : K$ *to denote* $\Gamma \vdash M : A \ \wedge\ \Gamma \vdash A \leq B \ \wedge\ \Gamma \vdash A, B : K$

THEOREM 7.1 (Soundness of algorithmic system of $\lambda\Pi^{\&}$). *For all* Γ, A, K, M,

1. $\Gamma \vdash_{\mathscr{A}} K \quad\Rightarrow\quad \Gamma \vdash K$
2. $\Gamma \vdash \star \ \wedge\ \Gamma \vdash_{\mathscr{A}} A : K \quad\Rightarrow\quad \Gamma \vdash A : K$
3. $\Gamma \vdash \star \ \wedge\ \Gamma \vdash_{\mathscr{A}} M : A \quad\Rightarrow\quad \Gamma \vdash M : A$

Proof. Simultaneously by induction on the depth of the derivation in the algorithmic system.

CASE (AT-&). Suppose that $\Gamma \vdash \star$ and that the derivation ends by an application of the rule AT-&. We have:

$$\begin{array}{ll}
\Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star & \Rightarrow\ \Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star \\
 & \Rightarrow\ \Gamma \vdash \pi x{:}A_{n+1}.B_{n+1} : \star \\
\Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n} : \star & \Rightarrow\ \Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\leq n} : \star \\
\Gamma \vdash_{\mathscr{A}} M : W_1 & \Rightarrow\ \Gamma \vdash M : W_1 \\
 & \Rightarrow\ \Gamma \vdash W_1 : \star \\
\Gamma \vdash M : W_1 \leq \{\pi x{:}A_i.B_i\}_{i\leq n} : \star & \Rightarrow\ \Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i\leq n} \\
\Gamma \vdash_{\mathscr{A}} N : W_2 & \Rightarrow\ \Gamma \vdash N : W_2 \\
 & \Rightarrow\ \Gamma \vdash W_2 : \star \\
\Gamma \vdash N : W_2 \leq \pi x{:}A_{n+1}.B_{n+1} : \star & \Rightarrow\ \Gamma \vdash N : \pi x{:}A_{n+1}.B_{n+1}
\end{array}$$

So we have a derivation ending by

$$\frac{\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i\leq n} \quad \Gamma \vdash N : \pi x{:}A_{n+1}.B_{n+1} \quad \Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star}{\Gamma \vdash M \&^{\{\pi x:A_i.B_i\}_{i\leq n+1}} N : \{\pi x{:}A_i.B_i\}_{i\leq n+1}}\ \text{T-\&}$$

CASE (AT-OAPP). Suppose that $\Gamma \vdash \star$ and that the derivation ends by an application of the rule AT-OAPP. We have:

$$
\begin{array}{ll}
\Gamma \vdash_{\mathscr{A}} M : W \leq_{\pi lub} \{\pi x{:}A_i.B_i\}_{i\leq n+1} & \Rightarrow \ \Gamma \vdash M : W \ \wedge \ \Gamma \vdash W \leq \{\pi x{:}A_i.B_i\}_{i\leq n+1} \\
 & \Rightarrow \ \Gamma \vdash W : \star \ \wedge \ \forall i \leq n+1.\ \Gamma \vdash A_i : \star \\
\Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star & \Rightarrow \ \Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star \\
\Gamma \vdash M : W \leq \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star & \Rightarrow \ \Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i\leq n+1} \\
\Gamma \vdash_{\mathscr{A}} N : A & \Rightarrow \ \Gamma \vdash N : A \\
 & \Rightarrow \ \Gamma \vdash A : \star \\
A_j = \min_{i\leq n+1}\{A_i \mid \Gamma \vdash A \leq A_i\} & \Rightarrow \ \Gamma \vdash A \leq A_j \\
\Gamma \vdash N : A \leq A_j : \star & \Rightarrow \ \Gamma \vdash N : A_j
\end{array}
$$

So we have a derivation ending by

$$
\frac{\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i\leq n+1} \quad \Gamma \vdash N : A_j}{\Gamma \vdash M \bullet N : B_j[x := N]} \ \text{T-OVER}
$$

Other cases are similar to those for $\lambda\Pi_{\leq}$. □

Again we have the completeness of the algorithmic system:

THEOREM 7.2 (Completeness of algorithmic system of $\lambda\Pi^{\&}$).

1. $\Gamma \vdash K \quad \Rightarrow \ \Gamma \vdash_{\mathscr{A}} K$
2. $\Gamma \vdash A : K \ \Rightarrow \ \exists K_a\ s.t.\ \Gamma \vdash_{\mathscr{A}} A : K_a \ \wedge \ K_a =_\beta K \ \wedge \ \Gamma \vdash K_a$
3. $\Gamma \vdash M : A \ \Rightarrow \ \exists A_a\ s.t.\ \Gamma \vdash_{\mathscr{A}} M : A_a \ \wedge \ \Gamma \vdash A_a \leq A$

Proof. Simultaneously by induction on the depth of the derivation in the original system.

CASE (T-&). Suppose that the derivation ended by an application of the rule T-&:

$$
\text{T-\&} \quad \frac{\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i\leq n} \quad \Gamma \vdash N : \pi x{:}A_{n+1}.B_{n+1} \quad \Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star}{\Gamma \vdash M \&^{\{\pi x:A_i.B_i\}_{i\leq n+1}} N : \{\pi x{:}A_i.B_i\}_{i\leq n+1}}
$$

Then

$$
\begin{array}{lll}
\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i\leq n} & \Rightarrow \ \exists W_1.\ \Gamma \vdash_{\mathscr{A}} M : W_1 \leq \{\pi x{:}A_i.B_i\}_{i\leq n} & IH \\
\Gamma \vdash N : \pi x{:}A_{n+1}.B_{n+1} & \Rightarrow \ \exists W_2.\ \Gamma \vdash_{\mathscr{A}} N : W_2 \leq \pi x{:}A_{n+1}.B_{n+1} & IH \\
\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i\leq n} & \Rightarrow \ \Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\leq n} : \star & \\
 & \Rightarrow \ \Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n} : \star & \\
\Gamma \vdash \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star & \Rightarrow \ \Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star &
\end{array}
$$

So we have a derivation ending by

$$
\text{AT-\&} \quad \frac{\begin{array}{cc}\Gamma \vdash_{\mathscr{A}} M : W_1 \leq \{\pi x{:}A_i.B_i\}_{i\leq n} & \Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n} : \star \\ \Gamma \vdash_{\mathscr{A}} N : W_2 \leq \pi x{:}A_{n+1}.B_{n+1} & \Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i\leq n+1} : \star\end{array}}{\Gamma \vdash_{\mathscr{A}} M \&^{\{\pi x:A_i.B_i\}_{i\leq n+1}} N : \{\pi x{:}A_i.B_i\}_{i\leq n+1}}
$$

Let $A_a = \{\pi x{:}A_i.B_i\}_{i\leq n+1}$, the result follows from the reflexivity of subtyping.

CASE (T-OAPP). Suppose that the derivation ended by an application of the rule T-OAPP:

T-OAPP
$$\frac{\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i\leq n} \qquad \Gamma \vdash N : A_i}{\Gamma \vdash M \bullet N : B_i[x := N]}$$

$$\begin{array}{lll}
\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i\leq n} & \Rightarrow & \Gamma \vdash_{\mathscr{A}} M : W \leq \{\pi x{:}A_i.B_i\}_{i\leq n} \\
 & \Rightarrow & \Gamma \vdash W \leq_{\pi lub} \{\pi x{:}A'_i.B'_i\}_{i\leq m} \\
 & & \quad \wedge\, \Gamma \vdash \{\pi x{:}A'_i.B'_i\}_{i\leq m} \leq \{\pi x{:}A_i.B_i\}_{i\leq n} \\
\Gamma \vdash_{\mathscr{A}} M : W & \Rightarrow & \Gamma \vdash M : W \\
 & \Rightarrow & \Gamma \vdash W : \star \\
 & \Rightarrow & \Gamma \vdash \{\pi x{:}A'_i.B'_i\}_{i\leq m} : \star \\
\Gamma \vdash N : A_i & \Rightarrow & \Gamma \vdash_{\mathscr{A}} N : A \leq A_i
\end{array}$$

So we have a derivation ending by

AT-OAPP
$$\frac{\begin{array}{c}\Gamma \vdash_{\mathscr{A}} M : W \leq_{\pi lub} \{\pi x{:}A'_i.B'_i\}_{i\leq m} \qquad \Gamma \vdash_{\mathscr{A}} \{\pi x{:}A'_i.B'_i\}_{i\leq m} : \star \\ \Gamma \vdash_{\mathscr{A}} N : A \qquad A'_j = \min_{i\leq m}\{A'_i \mid \Gamma \vdash A \leq A'_i\}\end{array}}{\Gamma \vdash_{\mathscr{A}} M \bullet N : B'_j[x := N]}$$

We continue as follows:

$$\begin{array}{lll}
\multicolumn{3}{l}{\Gamma \vdash \{\pi x{:}A'_i.B'_i\}_{i\leq m} \leq \{\pi x{:}A_i.B_i\}_{i\leq n}} \\
\quad\Rightarrow & \exists h.\, \Gamma \vdash \pi x{:}A'_h.B'_h \leq \pi x{:}A_i.B_i & \text{S-OVER} \\
\quad\Rightarrow & \Gamma \vdash A_i \leq A'_h \ \wedge\ \Gamma, x : A_i \vdash B'_h \leq B_i & \textit{generation of typing} \\
\quad\Rightarrow & \Gamma \vdash A \leq A'_h \ \wedge\ \Gamma, x : A \vdash B'_h \leq B_i & \Gamma \vdash A \leq A_i \\
\quad\Rightarrow & \Gamma \vdash A'_j \leq A'_h & A'_j = \min_{i\leq m}\{A'_i \mid \Gamma \vdash A \leq A'_i\} \\
\quad\Rightarrow & \Gamma, x : A'_j \vdash B'_j \leq B'_h & \textit{covariance} \\
\quad\Rightarrow & \Gamma, x : A \vdash B'_j \leq B'_h & \Gamma \vdash A \leq A'_j \\
\quad\Rightarrow & \Gamma, x : A \vdash B'_j \leq B_i & \textit{transitivity}
\end{array}$$

$\Gamma \vdash_{\mathscr{A}} N : A \Rightarrow \Gamma \vdash N : A$

$\Gamma \vdash N : A \ \wedge\ \Gamma, x : A \vdash B'_j \leq B_i \Rightarrow \Gamma \vdash B'_j[x := N] \leq B_i[x := N]$

Therefore, $\Gamma \vdash_{\mathscr{A}} M \bullet N : B'_j[x := N] \leq B_i[x := N]$, that is the result.

Other cases are similar to those in $\lambda\Pi_{\leq}$. $\square$

By an argument similar to the one for $\lambda\Pi_{\leq}$, we can prove the minimal typing property for $\lambda\Pi^{\&}$, that is that whenever $\Gamma \vdash_{\mathscr{A}} M : A$, then A is a minimal type of the term M.

Finally, note that the rules we added to the algorithmic system of $\lambda\Pi_{\leq}$ do not affect the termination of the algorithm, provided that the corresponding β conversion is decidable. Therefore from the above results we can also conclude that any subsystem of $\lambda\Pi^{\&}$ in which β-conversion is decidable, context formation, kinding, typing, and subtyping are decidable too.

8. CONCLUSION

In this work we have presented how to merge into a unique formalism dependent types, subtyping, and late bound overloading. The logical system we obtained is inevitably rather complex, but also very expressive.

The combination of subtyping with first order types does not need to be further justified since its need is acknowledged by several articles in the literature whose references are given in Section 2.1. The same papers show that some amount of overloading is necessary, as well. Up to now, this need was partially satisfied by the use of intersection types, which implement a very limited form of overloading.

The lack of elegant theories and, more generally, of studies of overloading may explain, if not justify, the use of intersection types as an ersatz of overloading. However, it is not very difficult to add overloaded functions to first order types, once we dispose of a complete theory of overloading. As a matter of fact, the relative complexity of $\lambda\Pi^{\&}$ does not come from the use of overloading but from the use of late binding. It is late binding that requires uneasy conditions on the kinding of types, conditions that enforce the circularity among kinding, typing and subtyping. Therefore, it is because of late binding that we could not start from existing systems of dependent types and subtyping, but we had to develop a brand new formalization, $\lambda\Pi_{\leq}$, which because of its broken circularity is prone to extensions. More recent works of other authors seem to confirm that the techniques we first introduced in $\lambda\Pi_{\leq}$ are the good ones, as we explained in the excursus ending Section 6.3.

The reward of these efforts is a very powerful system that, thanks precisely to late binding, allows the same kind of modular and incremental programming that has been made popular by object-oriented languages.

In this article we developed the theoretical part of the system and studied a large amount of theoretical properties ranging from confluence to subject reduction, from conservativity to transitivity elimination, from normalization properties to decidability. It is now necessary to explore the practical applications of this system, by defining appropriate decidable subsystems and by embedding them in programming languages and the technology of theorem provers. In that perspective it will be necessary to explore derived systems in which the formation and rewriting rules of the overloaded types can be weakened.

Acknowledgments It is out of doubt that the greatest boost to the quality of this work was given by Kathleen Fisher and the other two anonymous referees of Information and Computation. It is rather unusual to see reports so accurate, precise, and pertinent, and we do thanks the three referees for the time they dedicated to our work. But it is the first time we experience that *all* the reports of an article are of such a good quality. And the merit of that goes to our editor, Mariangiola Dezani, whom we thank for such a wise choice. Giuseppe Longo also deserves our gratitude, for having patiently commented several (about five) different versions of this work.

REFERENCES

AC96a. M.-V. Aponte and G. Castagna. Programmation modulaire avec surcharge et liaison tardive. In *Journées Francophones des Langages Applicatifs*, Val Morin, Québec, Canada, 1996. In French.

AC96b. D. Aspinall and A. Compagnoni. Subtyping dependent types. In *Proc. 11th Annual Synposium on Logic in Computer Science, IEEE*, pages 86–97, 1996.

Asp95. D. Aspinall. Subtyping with singleton types. In *Proc. Computer Science Logic, CSL'94*, number 933 in Lecture Notes in Computer Science. Springer, 1995.

Bar84. H.P. Barendregt. *The Lambda Calculus Its Syntax and Semantics*. North-Holland, 1984. Revised edition.

Bar92. H. Barendregt. Lambda calculi with types. In *Handbook of Logic in Computer Science*, volume 2. Oxford Carendon Press, 1992.

BCDC83. H. Barendregt, M. Coppo, and M. Dezani-Ciancaglini. A filter lambda model and the completeness of type assignment. *J.Symbolic Logic*, 48:931–940, 1983.

CAB+86. R.L. Constable, S.F. Allen, H.M. Bromley, W.R. Cleaveland, J.F. Cremer, R.W. Harper, D.J. Howe, T.B. Knoblock, N.P. Mender, P. Panangaden, J.T. Sasaki, and S.F. Smith. *Implementing Mathematics in the NuPRL Proof Development System.* Prentice-Hall, 1986.

Car88. L. Cardelli. A semantics of multiple inheritance. *Information and Computation*, 76:138–164, 1988. A previous version can be found in Semantics of Data Types, LNCS 173, 51-67, Springer, 1984.

Cas95. G. Castagna. Covariance and contravariance: conflict without a cause. *ACM Transactions on Programming Languages and Systems*, 17(3):431–447, 1995.

Cas96. G. Castagna. Integration of parametric and "ad hoc" second order polymorphism in a calculus with subtyping. *Formal Aspects of Computing*, 8(3):247–293, 1996.

Cas97. G. Castagna. *Object-Oriented Programming: A Unified Foundation*. Progress in Theoretical Computer Science. Birkäuser, Boston, 1997. ISBN 3-7643-3905-5.

CD80. M. Coppo and M. Dezani-Ciancaglini. An extension of the basic functionality theory for the λ-calculus. *Notre-Dame Journal of Formal Logic*, 21(4):685–693, October 1980.

CG97. A. Compagnoni and H. Goguen. Typed operational semantics for higher order subtyping. Technical Report ECS-LFCS-97-361, LFCS University of Edinburgh, July 1997.

CGL93. G. Castagna, G. Ghelli, and G. Longo. A semantics for λ&-*early*: a calculus with overloading and early binding. In M. Bezem and J.F. Groote, editors, *International Conference on Typed Lambda Calculi and Applications*, number 664 in Lecture Notes in Computer Science, pages 107–123, Utrecht, The Netherlands, March 1993. Springer.

CGL95. G. Castagna, G. Ghelli, and G. Longo. A calculus for overloaded functions with subtyping. *Information and Computation*, 117(1):115–135, 15 February 1995.

Cha92. C. Chambers. Object-oriented multi-methods in Cecil. In *ECOOP'92*, number 615 in Lecture Notes in Computer Science. Springer, 1992.

Che96. G. Chen. Dependent type system with subtyping. Technical Report LIENS-96-27, Laboratoire d'Informatique, Ecole Normale Supérieure - Paris, December 1996. Complete version available on `http://www.dmi.ens.fr/~gang/public_html/publi.html`.

Che97. G. Chen. Subtyping calculus of construction, extended abstract. In *22nd International Symposium on Mathematical Foundation of Computer Science*, volume 1295 of *Lecture Notes in Computer Science*. Springer, August 1997.

Che98. G. Chen. *Subtyping, type conversions and elimination of transitivity*. PhD thesis, Université Paris 7, December 1998.

Coq92. T. Coquand. Pattern matching with dependent types. In *Proceedings on Types for Proofs and Programs*, pages 71–83, 1992.

Cou97. J. Courant. A module calculus for pure type systems. In *Typed Lambda Calculus and Applications*, number 1210 in Lecture Notes in Computer Science, pages 112–128, 1997.

DG87. L.G. DeMichiel and R.P. Gabriel. Common Lisp Object System overview. In Bézivin, Hullot, Cointe, and Lieberman, editors, *Proc. of ECOOP '87 European Conference on Object-Oriented Programming*, number 276 in Lecture Notes in Computer Science, pages 151–170, Paris, France, June 1987. Springer.

Ghe93. G. Ghelli. Recursive types are not conservative over $F_{\leq}$. In M. Bezem and J.F. Groote, editors, *International Conference on Typed Lambda Calculi and Applications*, number 664 in Lecture Notes in Computer Science, pages 146–162, Utrecht, The Netherlands, March 1993. Springer. TLCA'93.

Gir87. J.Y. Girard. Linear logic. In *Theoretical Computer Science*, number 50, pages 1–102. North Holland, 1987.

HHP93. R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *Journal of the ACM*, 40(1):143–184, January 1993.

Hic95. J. J. Hickey. Formal abstract data types, December 1995. Unpublished draft.

Hin64. R. Hindley. The Church-Rosser property and a result of combinatory logic. Dissertation, 1964. University of Newcastle-upon-Tyne.

HL94. R. Harper and M. Lillibridge. A type-theoretic approach to higher-order modules with sharing. In *21st Annual Symposium on Principles Of Programming Languages*, pages 123–137, Portland, Oregon, January 1994. ACM Press.

Ler94. X. Leroy. Manifest types, modules, and separate compilation. In *Proceedings of the 21st Symposium on Principles of Programming Languages*, pages 109–122. ACM Press, January 1994.

Luo96. Z. Luo. Coercive subtyping in type theory. In *CSL'96, the 1996 Annual Conference of the European Association for Computer Science Logic, Utrecht*, volume 1258 of *Lecture Notes in Computer Science*, 1996.

Luo97. Z. Luo. Coercive subtyping. *Journal of Logic and Computation*, 9(13), 1997.

MH88. J.C. Mitchell and R. Harper. The essence of ML. *15th Ann. ACM Symp. on Principles of Programming Languages*, January 1988.

MQ85. D.B. Mac Queen. Modules for standard ML. *Polymorphism*, 2 (2), 1985.

MQ86. D.B. Mac Queen. Using dependent types to express modular structure: experience with Pebble and ML. In *Proc. 13th Annual ACM Symp. on Principles of Programming Languages*, 1986.

MTH90. R. Milner, M. Tofte, and R. Harper. *The Definition of Standard ML*. The MIT Press, 1990.

Pfe93. F. Pfenning. Refinement types for logical frameworks. In *Informal Proceedings of the 1993 Workshop on Types for Proofs and Programs*, May 1993.

Ros73. B. K. Rosen. Tree manipulation systems and Church-Rosser theorems. *Journal of the ACM*, 20:160–187, 1973.

Tai67. W.W. Tait. Intensional interpretation of functional of finite type I. *Journal of Symbolic Logic*, 32:198–212, 1967.

Zwa99. J. Zwanenburg. Pure type systems with subtyping. In *TLCA'99*, number 1581 in Lecture Notes in Computer Science. Springer, May 1999.

APPENDIX

A.1. $\lambda\Pi^{\&}$ SYSTEM

A.1.1. Typing and subtyping rules

A.1.1.1. Formation for Kind and Context

F-EMPTY
$$\frac{}{<>\vdash\ \star}$$

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

F-TYPE
$$\frac{\Gamma \vdash K \qquad \alpha \notin Dom(\Gamma)}{\Gamma, \alpha : K \vdash \star}$$

F-SUBTYPE
$$\frac{\Gamma \vdash A : K \qquad \alpha \notin Dom(\Gamma)}{\Gamma, \alpha \leq A : K \vdash \star}$$

F-Π
$$\frac{\Gamma, x : A \vdash K}{\Gamma \vdash \Pi x{:}A.K}$$

A.1.1.2. Kinding rules

K-VAR
$$\frac{\Gamma \vdash \star \qquad \alpha \in Dom(\Gamma)}{\Gamma \vdash \alpha : Kind_{\Gamma}(\alpha)}$$

K-π
$$\frac{\Gamma, x : A \vdash B : \star}{\Gamma \vdash \pi x{:}A.B : \star}$$

K-Λ
$$\frac{\Gamma, x : A \vdash B : K}{\Gamma \vdash \Lambda x{:}A.B : \Pi x{:}A.K}$$

K-APP
$$\frac{\Gamma \vdash A : \Pi x{:}B.K \qquad \Gamma \vdash M : B}{\Gamma \vdash AM : K[x := M]}$$

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

K-OVER
$$\frac{\begin{array}{c}\Gamma \vdash \star \qquad \forall i \in I\ .\ \Gamma \vdash \pi x{:}A_i.B_i : \star \\ \forall i \in I\ .\ \pi x{:}A_i.B_i \text{ is closed and in normal form} \\ \forall i, j \in I\ .\ \Gamma \vdash A_i \leq A_j \Rightarrow \Gamma, x : A_i \vdash B_i \leq B_j \\ \forall A.Fv(A) \subseteq Dom(\mathit{\Gamma}) \Rightarrow\ ((\forall i \in I.\ \Gamma \not\vdash A \leq A_i) \vee \\ (\exists! i \in I.\ \Gamma \vdash A \leq A_i \wedge \forall j \in I\ \Gamma \vdash A \leq A_j \Rightarrow \Gamma \vdash A_i \leq A_j))\end{array}}{\Gamma \vdash \{\pi x{:}A_i.B_i\}_{i \in I} : \star}$$

A.1.1.3. Typing rules

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

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

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

T-ε
$$\frac{\Gamma \vdash \star}{\Gamma \vdash \varepsilon : \{\}}$$

T-&
$$\frac{\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i \leq n} \quad \Gamma \vdash N : \pi x{:}A_{n+1}.B_{n+1} \quad \Gamma \vdash \{\pi x{:}A_i.B_i\}_{i \leq n+1} : \star}{\Gamma \vdash M \&^{\{\pi x:A_i.B_i\}_{i \leq n+1}} N : \{\pi x{:}A_i.B_i\}_{i \leq n+1}}$$

T-OAPP
$$\frac{\Gamma \vdash M : \{\pi x{:}A_i.B_i\}_{i \leq n} \qquad \Gamma \vdash N : A_i}{\Gamma \vdash M \bullet N : B_i[x := N]}$$

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

A.1.1.4. Subtyping rules

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

S-ApR
$$\frac{M_1 =_\beta M_1' \ \cdots \ M_n =_\beta M_n'}{\Gamma \vdash \alpha M_1 \cdots M_n \leq \alpha M_1' \cdots M_n'}$$

S-ApT
$$\frac{\Gamma \vdash \Gamma(\alpha) M_1..M_n \leq A}{\Gamma \vdash \alpha M_1..M_n \leq A}$$

S-ApSL
$$\frac{\Gamma \vdash B[x := M_1] M_2..M_n \leq C}{\Gamma \vdash (\Lambda x{:}A.B) M_1..M_n \leq C}$$

S-ApSR
$$\frac{\Gamma \vdash C \leq B[x := M_1] M_2..M_n}{\Gamma \vdash C \leq (\Lambda x{:}A.B) M_1..M_n}$$

S-OVER
$$\frac{\forall j \in J \, \exists i \in I \qquad \Gamma \vdash \pi x{:}A_i.B_i \leq \pi y{:}C_j.D_j}{\Gamma \vdash \{\pi x{:}A_i.B_i\}_{i \in I} \leq \{\pi y{:}C_j.D_j\}_{j \in J}}$$

A.1.2. Reduction

The β-conversion is given by context closure of the union of the following three notions of reduction:

$$(\lambda x{:}A.M)N \rightarrow_{\beta_1} M[x := N]$$
$$(\Lambda x{:}A.B)N \rightarrow_{\beta_2} B[x := N]$$

The $\beta^{\&}$-reduction in a context Γ is defined as follows:

If 1. N is closed and in normal form,

2. there exists $i \in [1..n]$ s.t. $\Gamma \vdash N{:}\, A_i$ and $\forall j \in [1..n] \ \Gamma \vdash N{:}\, A_j \ \Rightarrow \ \Gamma \vdash A_i \leq A_j$

then

$$(M_1 \&^{\{\pi x:A_h.B_h\}_{h=1..n}} M_2) \bullet N \rightarrow_{\beta^{\&}} \begin{cases} M_1 \bullet N & \text{for } i < n \\ M_2 \cdot N & \text{for } i = n \end{cases}$$

A.1.3. Algorithmic Rules for $\lambda\Pi^{\&}$

A.1.3.1. Algorithmic Subtyping Rules

AS-π
$$\frac{\Gamma \vdash_{\mathscr{A}} A' \leq A \qquad \Gamma, x : A' \vdash_{\mathscr{A}} B \leq B'}{\Gamma \vdash_{\mathscr{A}} \pi x{:}A.B \leq \pi x{:}A'.B'}$$

AS-ApR
$$\frac{M_1 =_\beta M_1' \ \cdots \ M_n =_\beta M_n'}{\Gamma \vdash_{\mathscr{A}} \alpha M_1 \cdots M_n \leq \alpha M_1' \cdots M_n'}$$

AS-ApT
$$\frac{\Gamma \vdash_{\mathscr{A}} \Gamma(\alpha) M_1..M_n \leq A}{\Gamma \vdash_{\mathscr{A}} \alpha M_1..M_n \leq A} \qquad A \not\equiv \alpha M_1' \ldots M_n'$$

AS-ApSL
$$\frac{\Gamma \vdash_{\mathscr{A}} B[x := M_1] M_2..M_n \leq C}{\Gamma \vdash_{\mathscr{A}} (\Lambda x{:}A.B) M_1..M_n \leq C}$$

AS-ApSR
$$\frac{\Gamma \vdash_{\mathscr{A}} C \leq B[x := M_1] M_2..M_n}{\Gamma \vdash_{\mathscr{A}} C \leq (\Lambda x{:}A.B) M_1..M_n} \qquad \begin{array}{c} C \not\equiv (\Lambda x{:}A'.B') M_1'..M_m' \\ C \not\equiv \alpha M_1' \ldots M_m' \end{array}$$

AS-OVER
$$\frac{\forall j \in J\ \exists i \in I \qquad \Gamma \vdash_{\mathscr{A}} \pi x{:}A_i.B_i \leq \pi y{:}C_j.D_j}{\Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i \in I} \leq \{\pi y{:}C_j.D_j\}_{j \in J}}$$

A.1.3.2. πlub Rules:

Lub-REFL
$$\frac{}{\Gamma \vdash_{\mathscr{A}} \pi x{:}A.B \leq_{\pi lub} \pi x{:}A.B}$$

Lub-ApT
$$\frac{\Gamma \vdash_{\mathscr{A}} \Gamma(\alpha) M_1..M_n \leq_{\pi lub} A}{\Gamma \vdash_{\mathscr{A}} \alpha M_1..M_n \leq_{\pi lub} A}$$

Lub-ApSL
$$\frac{\Gamma \vdash_{\mathscr{A}} B[x := M_1] M_2..M_n \leq_{\pi lub} C}{\Gamma \vdash_{\mathscr{A}} (\Lambda x{:}A.B) M_1..M_n \leq_{\pi lub} C}$$

Lub-OVER
$$\frac{}{\Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i \leq n} \leq_{\pi lub} \{\pi x{:}A_i.B_i\}_{i \leq n}}$$

A.1.3.3. Algorithmic Context Formation Rules:

AF-EMPTY
$$\frac{}{<> \vdash_{\mathscr{A}} \star}$$

AF-TERM
$$\frac{\Gamma \vdash_{\mathscr{A}} \star \quad \Gamma \vdash_{\mathscr{A}} A : \star \quad x \notin \mathit{Dom}(\Gamma)}{\Gamma, x : A \vdash_{\mathscr{A}} \star}$$

AF-TYPE
$$\frac{\Gamma \vdash_{\mathscr{A}} K \quad \alpha \notin \mathit{Dom}(\Gamma)}{\Gamma, \alpha : K \vdash_{\mathscr{A}} \star}$$

AF-SUBTYPE
$$\frac{\Gamma \vdash_{\mathscr{A}} K \quad \Gamma \vdash_{\mathscr{A}} A : K' \quad K =_\beta K' \quad \alpha \notin \mathit{Dom}(\Gamma)}{\Gamma, \alpha \leq A : K \vdash_{\mathscr{A}} \star}$$

AF-Π
$$\frac{\Gamma, x : A \vdash_{\mathscr{A}} K}{\Gamma \vdash_{\mathscr{A}} \Pi x{:}A.K}$$

A.1.3.4. Algorithmic Kinding Rules:

AK-VAR $$\frac{\alpha \in \mathit{Dom}(\Gamma)}{\Gamma \vdash_{\mathscr{A}} \alpha : \mathit{Kind}_\Gamma(\alpha)}$$

AK-π $$\frac{\Gamma \vdash_{\mathscr{A}} A : \star \quad \Gamma, x : A \vdash_{\mathscr{A}} B : \star}{\Gamma \vdash_{\mathscr{A}} \pi x{:}A.B : \star}$$

AK-Λ $$\frac{\Gamma \vdash_{\mathscr{A}} A : \star \quad \Gamma, x : A \vdash_{\mathscr{A}} B : K}{\Gamma \vdash_{\mathscr{A}} \Lambda x{:}A.B : \Pi x{:}A.K}$$

AK-APP $$\frac{\Gamma \vdash_{\mathscr{A}} A : \Pi x{:}B.K \quad \Gamma \vdash_{\mathscr{A}} M : B' \quad \Gamma \vdash_{\mathscr{A}} B' \leq B}{\Gamma \vdash_{\mathscr{A}} AM : K[x := M]}$$

AK-OVER $$\frac{\begin{array}{c}\Gamma \vdash_{\mathscr{A}} \star \qquad \forall i \in I \,.\, \Gamma \vdash_{\mathscr{A}} \pi x{:}A_i.B_i : \star \\ \forall i \in I \,.\, \pi x{:}A_i.B_i \text{ is closed and in normal form} \\ \forall i,j \in I \,.\, \Gamma \vdash_{\mathscr{A}} A_i \leq A_j \Rightarrow \Gamma, x : A_i \vdash_{\mathscr{A}} B_i \leq B_j \\ \forall A.\mathit{Fv}(A) \subseteq \mathit{Dom}(\Gamma) \Rightarrow \ ((\forall i \in I.\, \Gamma \not\vdash_{\mathscr{A}} A \leq A_i) \vee \\ (\exists! i \in I.\, \Gamma \vdash_{\mathscr{A}} A \leq A_i \wedge \forall j \in I\, \Gamma \vdash_{\mathscr{A}} A \leq A_j \Rightarrow \Gamma \vdash_{\mathscr{A}} A_i \leq A_j))\end{array}}{\Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i \in I} : \star}$$

A.1.3.5. Algorithmic Typing Rules:

AT-VAR $$\frac{x \in \mathit{Dom}(\Gamma)}{\Gamma \vdash_{\mathscr{A}} x : \Gamma(x)}$$

AT-λ $$\frac{\Gamma \vdash_{\mathscr{A}} A : \star \quad \Gamma, x : A \vdash_{\mathscr{A}} M : B}{\Gamma \vdash_{\mathscr{A}} \lambda x{:}A.M : \pi x{:}A.B}$$

AT-APP $$\frac{\Gamma \vdash_{\mathscr{A}} M : A \quad \Gamma \vdash_{\mathscr{A}} A \leq_{\pi lub} \pi x : B.C \quad \Gamma \vdash_{\mathscr{A}} N : B' \quad \Gamma \vdash_{\mathscr{A}} B' \leq B}{\Gamma \vdash_{\mathscr{A}} MN : C[x := N]}$$

AT-ε $$\frac{\Gamma \vdash_{\mathscr{A}} \star}{\Gamma \vdash_{\mathscr{A}} \varepsilon : \{\}}$$

AT-& $$\frac{\begin{array}{cc}\Gamma \vdash_{\mathscr{A}} M : W_1 \leq \{\pi x{:}A_i.B_i\}_{i \leq n} & \Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i \leq n} : \star \\ \Gamma \vdash_{\mathscr{A}} N : W_2 \leq \pi x{:}A_{n+1}.B_{n+1} & \Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i \leq n+1} : \star\end{array}}{\Gamma \vdash_{\mathscr{A}} M \&^{\{\pi x:A_i.B_i\}_{i \leq n+1}} N : \{\pi x{:}A_i.B_i\}_{i \leq n+1}}$$

AT-OAPP $$\frac{\begin{array}{cc}\Gamma \vdash_{\mathscr{A}} M : W \leq_{\pi lub} \{\pi x{:}A_i.B_i\}_{i \leq n+1} & \Gamma \vdash_{\mathscr{A}} \{\pi x{:}A_i.B_i\}_{i \leq n+1} : \star \\ \Gamma \vdash_{\mathscr{A}} N : A & A_j = min_{i \leq n+1}\{A_i \mid \Gamma \vdash A \leq A_i\}\end{array}}{\Gamma \vdash_{\mathscr{A}} M \bullet N : B_j[x := N]}$$

A.2. OVERLOADED FUNCTORS

```
signature Item = sig  type item;
                      val isequal: item * item -> bool
                 end
```

```
signature Tree = sig  structure i: Item;
                      type 'a tree;
                      val empty: 'a tree;
                      val cons: 'a * 'a tree * 'a tree -> 'a tree;
                      ...
                 end

signature OrdItem = sig  type item;
                         val isequal: item * item -> bool;
                         val isless: item * item -> bool
                    end

signature OrdTree = sig structure i: OrdItem;
                        type 'a tree;
                        ...
                    end

signature Dict =
sig
  type key
  type 'a dict
  val empty : 'a dict
  val isnull: 'a dict -> bool
  val find: key * 'a dict -> 'a
  val insert : key * 'a * 'a dict -> 'a dict
 end

mkDict =
  functor(t:Tree): Dict =
    struct
      type key = t.i.item;
      type 'a dict = (key * 'a) t.tree
      val empty = t.empty;
      val isnull = t.isnull
      fun find (k,d) = if isnull(d) then raise Notfound
            else let (k',a) = t.root(d) in if t.i.isequal(k,k') then a ...
      fun insert (k,a,d) = ...
    end
  and                         (*>>> overloading: "and" stands for "&" <<<*)
  functor(t:OrdTree): Dict =
    struct
      type key = t.i.item
      type 'a dict = (key * 'a) t.tree
      ...
      fun find (k,d) = if isnull(d) then raise Notfound
          else let (k',a) = t.root(d) in
            if t.i.isless(k,k') then find (k,left(d)) else ...
      fun insert (k,a,d) = if isnull(d) then t.cons((k,a),empty,empty)
                           else (* Ordered search of a free position *)
  end
```