An optimized complete semi-algorithm for system $\mathbf{F}^{\eta}$

Christophe Raffalli
Université de Savoie
email: Christophe.Raffalli@univ-savoie.fr, tel: (33) 4 79 75 81 03

Mars 1999

Abstract

In this paper we give a new deterministic presentation of system F with η -reduction. This presentation allow us to write a complete semi-algorithm for this system that may be useful in a real programming language.

keywords: lambda-calculus, type-inference, type-checking

1 Introduction

Motivation Most of the statically typed programming language (SML, OCaml, Haskell, ...) are based on Milner's restriction [1] of Girard and Reynolds System F [4, 13]. To improve the language, some complex extensions of the type-system are added to handle the needed features (modules with abstract types, object, some kind of polymorphic recursion). These extensions are quite complex both at the theoretical and the programming level. However, they leads to a decidable type-inference algorithm.

Most, if not all, of these extensions could be handled inside system F. For instance, existential types are definable in system F and can be used to construct tuples with abstract types which correspond to the notion of structure or module. Then F polymorphism and the usual notion of function can be used to encode SML or OCaml functor.

The advantage of this approach is a smaller core language at the theoretical level. The main problem is the fact that the type system is undecidable [14, 15].

However, there may be a usable semi-algorithm if the "usual programs" do not really use the aspects of the system that make it undecidable. This is already the case for ML which is theoretically DEXPTIME [11] but these worst cases never appear in real programs.

Contribution of this paper The main contribution of this paper is a new semi-algorithm that may reach this goal.

We choose to study Mitchell's version of system F [10], because it is based on a subtyping relation and subtyping is really needed to have a powerful module system.

We give a new version of the subtyping for this system which has the property of being deterministic: there is only one applicable rule (in fact there may be two applicable rules, but in this case they commute!). Non-determinism is reduced to one rule when we need to instantiate a bound variable with an unknown formula. This will allow us to design a complete semi-algorithm to solve a system of subtyping judgments based on methods very similar to Huet's higher-order unification [6].

We also simplify the typing rules to show that all the instances of the subtyping rule can be limited to the axiom level. From this we deduce a typing algorithm that associates a system of subtyping judgments to a typing judgment. Our simplification leads to smaller system and this was really the main optimization that made the algorithm succeed on small polymorphic terms.

$$\frac{}{\Gamma, x : A \vdash_F x : A}\, Ax$$

$$\frac{\Gamma, x : A \vdash_F t : B}{\Gamma \vdash_F \lambda x\, t : A \to B} \to_i \qquad \frac{\Gamma \vdash_F t : A \to B \qquad \Gamma \vdash_F u : A}{\Gamma \vdash_F (t\ u) : B} \to_e$$

$$\frac{\Gamma \vdash_F t : A \qquad X \notin \mathcal{V}(\Gamma)}{\Gamma \vdash_F t : \forall X\, A} \forall_i \qquad \frac{\Gamma \vdash_F t : \forall X\, A}{\Gamma \vdash_F t : A[X \Leftarrow C]} \forall_e$$

Table 1: The rules for system **F**

Related work Our presentation of subtyping differs from Mitchell's [10] because it can handle the introduction of the quantification. This means that we do need to keep this rule in the type system. Moreover, in our system, the transitivity is only an admissible rule. This is also the case in Longo, Milsted and Soloviev work [9]. However, in their presentation there is still some non-determinism (the $\forall_{n\geq 0}$ rule and the arrow rule do not commute) and their subtyping still can not handle the introduction of quantification.

Giannini and Ronchi della Rocca [3] have studied a complete stratification (based on the distance between the quantifier and its variables) of Girard's System F .

In [12], Piperno and Ronchi della Rocca study another stratification for the system F^η. The algorithm proceed by trying to type larger and larger η-expansions of a term in a decidable restriction of F^η.

Our system uses a more direct approach: we get the algorithm from the rules because they are syntax directed. It also seems that experiments and optimizations have not been studied to check the practical interest of the previous algorithms while this was our main goal. However, these works are based on clear stratifications of the original calculus while we can not give a proper definition of the stratification underlying our algorithm.

One should also cite some work on the rank stratification (based on the depth of the quantifier), with an algorithm for the rank-2 [8] and the fact that this stratification is undecidable from Rank-3 [7].

Organisation of the paper After a few preliminaries, we define our presentation of system F with η-reduction and we establish the main lemmas we need and the proof of the equivalence of our system with Mitchell's (theorem 3.26). Then we present the simplified system and show its relation to the original system (proposition 4.1 and theorem 4.6). In section 5, we give the definition of the algorithm and we prove its correctness and completeness (theorem 6.5 and 6.12. In the next two sections we discuss briefly implementation and optimization issues and report our experimental results. We end the paper with some hints on further interesting work.

2 Notation and preliminaries on System F.

We consider only "Curry style" system **F** without any type information in terms. The set of λ-terms, denoted Λ, is constructed from a set of variables $\mathcal{V}_\Lambda$ using applications and abstractions. Applications are written $(t\ t_1\ \ldots\ t_n)$ and abstractions are written $\lambda x\, t$. We respectively denote the β-reduction, η-reduction and $\beta\eta$-reduction by $\triangleright_\beta$, $\triangleright_\eta$ and $\triangleright_{\beta\eta}$.

The set of formulas $\mathcal{F}$ is constructed from the set of predicate variables $\mathcal{V}_\mathcal{F}$ using implications and universal quantifications. Implications are written $A \to B$ and universal quantifications are written $\forall X\, F$ with the smallest possible scope: $\forall X\, F \to G$ means $(\forall X\, F) \to G$. $\mathcal{V}(A)$ will denote the set of all free propositional variables in A. If $\Gamma = x_1 : A_1, \ldots, x_n : A_n$ is a typing context, $\mathcal{V}(\Gamma)$ will denote the set of free variables in $A_1, \ldots, A_n$.

$$\frac{X \in \mathcal{V}_{\mathcal{F}}}{X \subset_\Gamma X}\, Ax^{\subset} \qquad \frac{B \subset_{C,\Gamma} D \qquad C \subset_{C,\Gamma} A}{A \to B \subset_\Gamma C \to D}\, \to^{\subset}$$

$$\frac{A[X \Leftarrow C] \subset_\Gamma B}{\forall X\, A \subset_\Gamma B}\, \forall_l^{\subset} \qquad \frac{A \subset_\Gamma B \qquad X \notin \mathcal{V}(\Gamma)}{A \subset_\Gamma \forall X\, B}\, \forall_r^{\subset}$$

Table 2: The sub-typing rules for system $\mathbf{F}^\eta$.

We write $[\chi \Leftarrow \phi]$ for the substitution of the variables χ by ϕ. We use this notation both for terms and formulas. If σ is a substitution, $F\sigma$ denotes the application of the substitution to the formula F. We only use substitution with "finite support": for all substitution σ, $X\sigma = X$ except for a finite number of variables.

The rules of system $\mathbf{F}$ are given in table 1. It is well known that both strong normalisation and subject-reduction (for β-reduction) holds in this system [5]. But, it is also clear that subject-reduction does not holds for η-reduction.

3 System $\mathbf{F}^\eta$.

System $\mathbf{F}^\eta$ is an extension of system $\mathbf{F}$ which was designed to get subject-reduction for η-reduction too. To achieve this goal, one solution is to use a subtyping relation to manipulate the quantification rules. This will enable some permutations of quantifications which were correct in system $\mathbf{F}$ but needed some η-expansions.

We give an unusual presentation of this subtyping to have a system as deterministic as possible. To do so we add in the subtyping relation $\subset_\Gamma$ on formulas a typing context Γ called the set of constraints. This subtyping is defined as the smallest relation derivable using the deduction rules given in table 2. We often omit the λ-variables in the set of constraints.

The meaning of this relation is the following: If we have $A \subset_\Gamma B$ with $\{X_1, \ldots, X_n\} \notin \Gamma$, then $\forall X_1 \ldots \forall X_n A$ is a subtype of B which means that any term belonging to the first type inhabits the second too. One should note that our subtyping is not transitive (the lemma 3.9 gives a weak version of the transitivity).

Properties of the sub-typing. We have chosen to prove directly that our presentation of the system $\mathbf{F}^\eta$ is equivalent to system $\mathbf{F}$ with the η-reduction rule. Similarly, we could have prove that our type system is equivalent to Mitchell's system. But this does not seem much easier: the same fundamental lemmas are needed and they are also used to prove the competeness of the simplified version of the system.

Lemma 3.1 *For any formula A and any typing context Γ we have $A \subset_\Gamma A$.*

Proof: Induction on A. ■

Lemma 3.2 *If $\Gamma \vdash_F t : A$, $A \subset_\Gamma B$ and $\mathcal{V}(\Gamma) \cap \Gamma = \emptyset$ then we can find t' an η-expansion of t ($t' \triangleright_\eta t$) such that $\Gamma \vdash_F t' : B$.*

Proof: The proof is by induction on the derivation of $A \subset_\Gamma B$. ■

Lemma 3.3 *If $A \subset_\Gamma B$ and if σ is a substitution such that $X\sigma = X$ for all $X \in \mathcal{V}(A) - \mathcal{V}(\Gamma) \cup \mathcal{V}(B)$ then we have $A\sigma \subset_{\Gamma\sigma} B\sigma$.*

Lemma 3.4 *If $A \subset_\Gamma B$ with $X \in \mathcal{V}(A) - \mathcal{V}(\Gamma) \cup \mathcal{V}(B)$ then for any variable Y such that $Y \notin \mathcal{V}(\Gamma) \cup \mathcal{V}(A) \cup \mathcal{V}(B)$ we have $A[X \Leftarrow Y] \subset_\Gamma B$.*

Proof: The proof of the two previous lemmas is done by simultaneous induction on the derivation of $A \subset_\Gamma B$. ■

Definition 3.5 *We define* $A \subset B$ *by* $A \subset_A B$

Lemma 3.6 *If* $A \subset_\Gamma B$ *and* $(\mathcal{V}(\Delta) \cup \mathcal{V}(B)) \cap \mathcal{V}(A) \subset (\mathcal{V}(\Gamma) \cup \mathcal{V}(B)) \cap \mathcal{V}(A)$ *then* $A \subset_\Delta B$

Proof: The proof is by induction on the derivation of $A \subset_\Gamma B$ using lemma 3.4 for the $\forall_l^\subset$ rule. ■

Corollary 3.7 $A \subset B$ *if and only if for any context* Γ *we have* $A \subset_\Gamma B$.

Proof: It is an immediate consequence of lemma 3.6. ■

Corollary 3.8 $A \subset_\Gamma B$ *if and only if* $A \subset_{\Gamma,B} B$.

Proof: Immediate from the lemma 3.6

Lemma 3.9 *If* $A \subset_\Gamma B$ *and* $B \subset_\Delta C$ *then there exists a substitution* σ *such that* $A\sigma \subset_{\Gamma\cup\Delta} C$, *and for all variable* $X \in \mathcal{V}(\Gamma) \cup \mathcal{V}(B)$, $X\sigma = X$.

Proof: It is an induction on the sum of the size of both proofs using lemmas 3.3 and 3.4 for implication rule and 3.4 for the quantification rules. ■

Corollary 3.10 *The relation* $\subset$ *is transitive.*

Proof: The lemma 3.9 gives that $A \subset B$ and $B \subset C$ implies $A\sigma \subset_{A,B} C$ and the condition on σ implies $A\sigma = A$, which gives the wanted result using lemma 3.6. ■

Definition 3.11 (Equivalence) *We define the relation* $\sim$ *as the smallest equivalence relation on formulas such that*

- $\forall X(A \to B) \sim A \to \forall X\, B$
- $A \sim A'$ *implies* $\forall X\, A \sim \forall X\, A'$
- $A \sim A'$ *and* $B \sim B'$ *implies* $A \to B \sim A' \to B'$

Definition 3.12 (Prenex form) *We define the prenex form of a formula* A *denoted* $\pi(A)$ *by induction as follows:*

- $\pi(X) = X$
- $\pi(\forall X\, A) = \forall X\, \pi(A)$
- $\pi(A \to B) = \forall X_1, \ldots, X_n(A \to B')$ *if* $\pi(B) = \forall X_1, \ldots, X_n B'$

Lemma 3.13 *For any formula* A, $A \sim \pi(A)$

Proof: trivial ■

Lemma 3.14 *For any formulas* A *and* B, $A \sim B$ *implies* $A \subset B$ *and* $B \subset A$

Proof: By induction on the definition of $A \sim B$ using lemma 3.10 (The converse is true, but we do not need it). ■

Lemma 3.15 *For any formulas* A, B *and* C *we have:*

- $A \sim B$ *and* $B \subset_\Gamma C$ *implies* $A \subset_\Gamma C$
- $A \subset_\Gamma B$ *and* $B \sim C$ *implies* $A \subset_\Gamma C$

Proof: The first proposition is proved using $A \subset B$ and the same argument than for corollary 3.10. For the second proposition, the proof is similar to the proof of lemma 3.9 (here no substitutions are needed). ■

$$\frac{}{\Gamma, x : A \vdash x : A}\, Ax$$

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x\, t : A \to B} \to_i \qquad \frac{\Gamma \vdash t : A \to B \qquad \Gamma \vdash u : A}{\Gamma \vdash (tu) : B} \to_e$$

$$\frac{\Gamma \vdash t : A \qquad A \subset_\Gamma B}{\Gamma \vdash t : B} \subset$$

Table 3: The typing rules for system $\mathbf{F}^\eta$.

Properties of the type-system. The typing rules of the system $\mathbf{F}^\eta$ are given in table 3.

Lemma 3.16 *If $\Gamma \vdash_F t : A$ then $\Gamma \vdash t : A$.*

Proof: It is easy to prove that $\forall X\, A \subset_\emptyset A[X \Leftarrow B]$ and $A \subset_{\{X\}} \forall X\, A$. So the $\subset$ rule can simulate both introduction and elimination of the universal quantification. From this we get an obvious proof of the result by induction on the structure of the proof of $\Gamma \vdash_F t : A$. ■

Lemma 3.17 *If $\Gamma \vdash t : F$ then we can find $t' \in \Lambda$ such that $t' \triangleright_\eta t$ and $\Gamma \vdash_F t' : F$.*

Proof: The proof is by induction on the structure of the derivation of $\Gamma \vdash t : F$. The only non-trivial case is the $\subset$ rule, but this case follows from lemma 3.2. ■

Theorem 3.18 *Strong normalisation holds for system $\mathbf{F}^\eta$.*

Proof: This result follows from lemma 3.17 and the strong normalisation of system $\mathbf{F}$[5]. ■

Definition 3.19 (size of proofs) *The size of a typing derivation is defined as the number of typing rules. The subtyping rules to derive judgment of the form $A \subset_\Gamma B$ are not counted (the subtyping rule $\subset$ itself counts).*

Lemma 3.20 *If $\Gamma \vdash t : A$, then for all substitution σ, $\Gamma\sigma \vdash t : A\sigma$. Moreover, the size of the proof is left unchanged.*

Proof: The proof is by induction on the derivation of $\Gamma \vdash t : A$. The only non trivial case is the case of the $\subset$ rule. In this case, we have $\Gamma \vdash t : B$, $B \subset_\Gamma A$. For each variable $X \in \mathcal{V}(B) - (\mathcal{V}(\Gamma) \cup \mathcal{V}(A)$, we choose a variable X' such that $X'\sigma = X'$. We apply the lemma 3.4 to get $B[X \Leftarrow X'] \subset_\Gamma A$. Then we can apply the induction hypothesis to get $\Gamma \vdash t : B[X \Leftarrow X']$. After doing that for each variable we get a formula B' such that $B' \subset_\Gamma A$, $\Gamma \vdash t : B'$ and $X\sigma = X$ for all the variables $X \in \mathcal{V}(B') - (\mathcal{V}(\Gamma) \cup \mathcal{V}(A)$.

Finally we can apply lemma 3.3 to find $B'\sigma \subset_{\Gamma\sigma} A\sigma$ and the induction hypothesis to find $\Gamma\sigma \vdash t : B'\sigma$. This gives the wanted result if we use the $\subset$ rule. The size of the proof is left unchanged because we only changed the subtyping derivations in the proof. ■

Lemma 3.21 *If $\Gamma \vdash t : A$, then there is a proof with the same conclusion sequent which do not use two successive instances of the $\subset$ rule.*

Proof: We can consider the following reduction rule for proofs: let us assume we have a sub-proof with the following shape:

$$\dfrac{\dfrac{\Gamma \vdash t : A \qquad A \subset_\Gamma B}{\Gamma \vdash t : B} \subset \qquad B \subset_\Gamma C}{\Gamma \vdash t : C} \subset$$

Then we can use lemma 3.9 to find σ such that $A\sigma \subset_\Gamma C$ and for all $X \in \Gamma$, $X\sigma = X$. So we have $\Gamma\sigma = \Gamma$ and we can apply lemma 3.20 to find $\Gamma \vdash t : A\sigma$. Then we can apply the $\subset$ rule to get $\Gamma \vdash t : C$. ■

Moreover, because applying the lemma 3.20 did not increase the size of the proof, the size of the resulting proof is smaller than the original proof. Therefore, we can apply this process indefinitely to any proof. When this process ends the final proof does not use two succesive instances of the $\subset$ rule. ■

Lemma 3.22 *Weakening is admissible in system* $\mathbf{F}^\eta$.

Proof: We have to prove that if $\Gamma \vdash t : A$ then $\Gamma, x : C \vdash t : A$. We prove this by induction on the derivation. Here again the only non trivial case is the case of the $\subset$ rule. In this case, we have $\Gamma \vdash t : B$ and $B \subset_\Gamma A$. We can apply the lemma 3.4 to rename all the variables $X \in (\mathcal{V}(B) \cap \mathcal{V}(C)) - (\mathcal{V}(\Gamma) \cup \mathcal{V}(A))$ and get $B' \subset_\Gamma A$ with $(\mathcal{V}(B') \cap \mathcal{V}(C)) - (\mathcal{V}(\Gamma) \cup \mathcal{V}(A)) = \emptyset$. The lemma 3.20 gives $\Gamma \vdash t : B'$ (without changing the size of the proof), the induction hypothesis gives $\Gamma, C \vdash t : B'$ and lemma 3.6 gives $B' \subset_{\Gamma,C} A$ which leads to the expected result using the $\subset$ rule. ■

Lemma 3.23 *If* $\Gamma, x : A \vdash t : B$ *and* $\Gamma \vdash u : A$ *then* $\Gamma \vdash t[x \Leftarrow u] : B$.

Proof: The proof is by induction on the derivation of $\Gamma, x : A \vdash t : B$ and uses lemma 3.22 for the axiom case. ■

Lemma 3.24 *If* $\Gamma, x : A \vdash t : B$ *and* x *does not occur in* t *then* $\Gamma \vdash t : B$.

Proof: The proof is by induction on the derivation. The condition x does not occur in t is used for the axiom case only. The case of the $\subset$ rule is obvious using lemma 3.6. ■

Theorem 3.25 *Subject reduction for* $\beta\eta$*-reduction holds in system* $\mathbf{F}^\eta$.

Proof: We want to prove that if $\Gamma \vdash t : A$ ans $t \triangleright_{\beta\eta} t'$ then $\Gamma \vdash t' : A$. The proof goes by induction on the length of the reduction and for each step, by induction on the structure of the term. The only non trivial cases are the redexes.

- In the case of a β-redex, using lemma 3.21, we have a proof of the following shape:

$$\dfrac{\dfrac{\dfrac{\vdots}{\Gamma, x : A \vdash t : B}}{\Gamma \vdash \lambda x\, t : A \to B}\to_i \qquad \dfrac{\vdots}{A \to B \subset_\Gamma C \to D}}{\dfrac{\Gamma \vdash \lambda x\, t : C \to D}{\Gamma \vdash (\lambda x\, t)\ u : D}\to_e} \subset \qquad \dfrac{\vdots}{\Gamma \vdash u : C}$$

  The proof of $A \to B \subset_\Gamma C \to D$ necessarely ends by the $\to^\subset$ rule. So we have $C \subset_{\Gamma,C} A$, $B \subset_{\Gamma,C} D$. Then by using the $\subset$ rule we find $\Gamma \vdash u : A$ from $\Gamma \vdash u : C$. Thus using lemma 3.23, we get $\Gamma \vdash t[x \Leftarrow u] : B$ from $\Gamma, x : A \vdash t : B$. Finally, by a last use of the $\subset$ rule, we find $\Gamma \vdash t[x \Leftarrow u] : D$.

- In the case of the η-redex, using lemma 3.21, we have a proof of the following shape with $\Gamma' = \Gamma, x : A$:

$$\dfrac{\dfrac{\dfrac{\dfrac{\vdots}{\Gamma' \vdash t : C \to D} \qquad \dfrac{\dfrac{}{\Gamma' \vdash x : A}Ax \qquad \dfrac{\vdots}{A \subset_{\Gamma'} C}}{\Gamma' \vdash x : C}\subset}{\Gamma' \vdash (t\ x) : D}\to_e \qquad \dfrac{\vdots}{D \subset_{\Gamma'} E}}{\Gamma' \vdash (t\ x) : E}\subset}{\Gamma \vdash \lambda x\, (t\ x) : A \to E}\to_i$$

$$\frac{A \subset B}{\Gamma, x : A \vdash_s x : B}\, Ax \qquad \frac{\Gamma, x : A \vdash_s t : B}{\Gamma \vdash_s \lambda x\, t : A \to B} \to_i$$

$$\frac{\Gamma \vdash_s t : (\forall X_1, \ldots, X_n\, A) \to B \qquad \Gamma \vdash u : A \qquad X_1, \ldots, X_n \notin \mathcal{V}(\Gamma)}{\Gamma \vdash_s (tu) : B} \to_e$$

Table 4: The simplified system.

We know that $\lambda x\,(t\ x) \rhd_\eta t$. So x does not appear free in t. Using lemma 3.24, we get $\Gamma \vdash t : C \to D$. Then we can produce the following proof to get the wanted result:

$$\frac{\begin{array}{c}\vdots \\ \Gamma \vdash t : C \to D\end{array} \qquad \frac{\begin{array}{c}\vdots \\ A \subset_{\Gamma'} C\end{array} \qquad \begin{array}{c}\vdots \\ D \subset_{\Gamma'} E\end{array}}{C \to D \subset_\Gamma A \to E} \to^{\subset}}{\Gamma \vdash t : A \to E} \subset$$

■

Theorem 3.26 *$\Gamma \vdash t : F$ iff exists $t' \in \Lambda$ such that $t' \rhd_\eta t$ and $\Gamma \vdash_F t' : F$.*

Proof: This lemma follows from lemma 3.16 and 3.17 and from theorem 3.25. ■

4 Simplification of the typing

We will show that we can restrict the $\subset$ rule to the axiom, just leaving some $\forall$ introduction for the right premice of the $\to_e$ rule and the conclusion. We write $\Gamma \vdash_s t : B$ for provable sequent in the simplified system. We give the rule of the simplified system in table 4

Proposition 4.1 (Correctness of the simplified system) *$\Gamma \vdash_s t : A$ and $A \subset_\Gamma A'$ implies $\Gamma \vdash t : A'$.*

Proof: Immediate: all the rules of the simplified system are derived rules of the complete system. ■

Lemma 4.2 *If $\Gamma, x : A \vdash_s t : B$ and $A' \subset A$ then $\Gamma, x : A' \vdash_s t : B$*

Proof: Immediate induction using lemma 3.10 for the axiom case. ■

Definition 4.3 *$A \simeq B$ defined by:*

- $X \simeq X$
- $A \to B \simeq A' \to B'$ *iff* $A \sim A'$ *and* $B \simeq B'$
- $\forall X\, A \simeq \forall X\, A'$ *iff* $A \simeq A'$

Lemma 4.4 *If $\Gamma \vdash_s t : A$, $\Gamma \sim \Gamma'$ and $A \simeq A'$ then $\Gamma \vdash_s t : A'$*

Proof: Induction on the proof of $\Gamma \vdash_s t : A$ using lemma 4.2 for the implication rule. ■

Lemma 4.5 *If $\Gamma \vdash_s t : A$, $A \subset_\Gamma B$, $\pi(B) = \forall X_1, \ldots, X_n\, B'$ and B' does not start with a quantifier then $\Gamma \vdash_s t : B'$*

Proof: We prove this by induction on the proof of $\Gamma \vdash_s t : A$.

- If the last rule is the axiom: $\Gamma, x : A' \vdash_s x : A$ with $A' \subset A$ and $A \subset_\Gamma B$. Using lemma 3.9, 3.14 and 3.13, we find $A' \subset_\Gamma \pi(B)$. We have $\pi(B) = \forall X_1, \ldots, X_n\, B'$ thus $A' \subset_\Gamma B'$, because the proof of $A' \subset_\Gamma \pi(B)$ can only ends by the $\forall_r^\subset$ of the $\forall_l^\subset$ rules and both rules commute. This gives $\Gamma, x : A' \vdash_s x : B'$.

- If the last rule is the implication introduction rule: We have $\Gamma, x : C \vdash_s t : A$ with $C \to A \subset_\Gamma B$. Using lemma 3.9, 3.14 and 3.13, we find $C \to A \subset_\Gamma \pi(B)$. We have $\pi(B) = \forall X_1, \ldots, X_n\, B'$ which implies $C \to A \subset_\Gamma B'$. Therefore, $B' = C' \to A'$ with $C' \subset C$ and $A \subset_\Gamma A'$ because the last rule can only be the $\to^\subset$ rule. By definition of the prenex form, we have $\pi(A) = \forall X_1, \ldots, X_n A'$, so the induction hypothesis gives $\Gamma, x : C \vdash_s t : A'$ which gives $\Gamma, x : C' \vdash_s t : A'$ using lemma 4.2. Therefore we have $\Gamma \vdash_s \lambda x\, t : B'$.

- If the last rule is the implication elimination rule: We have $\Gamma \vdash_s t : (\forall Y_1, \ldots, Y_p\, C) \to A$, $\Gamma \vdash_s u : C$, $Y_1, \ldots, Y_p$ not free in Γ and $A \subset_\Gamma B$. From $\pi(B) = \forall X_1, \ldots, X_n\, B'$ we get $(\forall Y_1, \ldots, Y_p\, C) \to A \subset_\Gamma (\forall Y_1, \ldots, Y_p\, C) \to B$ and $\pi((\forall Y_1, \ldots, Y_p\, C) \to B) = \forall X_1, \ldots, X_n((\forall Y_1, \ldots, Y_p\, C) \to B')$. Therefore the induction hypothesis gives $\Gamma \vdash_s t : (\forall Y_1, \ldots, Y_p\, C) \to B'$ which implies $\Gamma \vdash_s (t\, u) : B'$. ■

Theorem 4.6 (Completeness of the simplified system) *$\Gamma \vdash t : A$ with $\pi(A) = \forall X_1, \ldots, X_n\, A'$, A' not starting with a quantifier implies $\Gamma \vdash_s t : A'$.*

Proof: We prove this by induction on the proof of $\Gamma \vdash_s t : A$.

- For the axiom case we have $\Gamma, x : A \vdash x : A$ with $\pi(A) = \forall X_1, \ldots, X_n\, A'$. Thus we have $A \subset \forall X_1, \ldots, X_n\, A'$. By the same argument than in the axiom case of the previous proof we find $A \subset A'$. Therefore we have $\Gamma, x : A \vdash_s x : A'$.

- For the implication introduction case, we have $\Gamma, x : A \vdash t : B$ with $\pi(A \to B) = \forall X_1, \ldots, X_n\, C$. Thus we have $C = A \to B'$ with $\pi(B) = \forall X_1, \ldots, X_n\, B'$. By induction hypothesis we find $\Gamma, x : A \vdash_s t : B'$ which gives $\Gamma \vdash_s \lambda x\, t : C$.

- For the implication elimination case, we have $\Gamma \vdash t : B \to A$, $\Gamma \vdash u : B$ and $\pi(A) = \forall X_1, \ldots, X_n\, A'$. We have $\pi(B \to A) = \forall X_1, \ldots, X_n\, (B \to A')$ and the induction hypothesis gives $\Gamma \vdash_s t : B \to A'$. Let B' be the formula such that $\pi(B) = \forall Y_1, \ldots, Y_p\, B'$. We have by induction hypothesis $\Gamma \vdash_s u : B'$. Moreover, we have $B \to A' \simeq \forall Y_1, \ldots, Y_p\, B' \to A'$. Therefore using lemma 4.4 we find $\Gamma \vdash_s t : \forall Y_1, \ldots, Y_p\, B' \to A'$. $Y_1, \ldots, Y_n$ were bound variables of B, they can be chosen not free in Γ, so we have $\Gamma \vdash_s (t\, u) : A'$.

- For the subtyping rule, we have $\Gamma \vdash t : B$, $B \subset_\Gamma A$ and $\pi(A) = \forall X_1, \ldots, X_n\, A'$. Let B' be the formula such that $\pi(B) = \forall Y_1, \ldots, Y_p\, B'$. We have by induction hypothesis $\Gamma \vdash_s u : B'$. By lemmas 3.13 and 3.15 we get $\forall Y_1, \ldots, Y_p\, B' \subset_\Gamma A$ which implies $B' \subset_\Gamma A$. Then, the lemma 4.5 gives $\Gamma \vdash_s u : A'$. ■

5 Definition of the algorithm

Definition 5.1 (Formula with meta-variables) *We consider that the set $\mathcal{V}_\mathcal{F}$ used to define formulas was defined as $\mathcal{V}'_\mathcal{F} \times \mathbb{N}$, that is each element X of $\mathcal{V}_\mathcal{F}$ was of the form Y^i for $Y \in \mathcal{V}'_\mathcal{F}$ and $i \in \mathbb{N}$. We choose for each natural $n \neq 0$ a set of meta-variables of arity n, $\mathcal{M}_n$. We define the set of formulas with meta-variables $\mathcal{F}^\mathcal{M}$ as the smallest set such that:*

- $X^i \in \mathcal{F}^\mathcal{M}$ *if* $X \in \mathcal{V}'_\mathcal{F}$ *and* $i \in \mathbb{N}$
- $\alpha^i(\chi_1, \ldots, \chi_n)$ *if* $\alpha \in \mathcal{M}_n$, $i, n \in \mathbb{N}$ *and* $\chi_1, \ldots, \chi_n \in \mathcal{V}'_\mathcal{F} \cup \mathcal{M}_1$
- $A \to B \in \mathcal{F}^\mathcal{M}$ *if* $A, B \in \mathcal{F}^\mathcal{M}$

- $\forall X\, A \in \mathcal{F}^{\mathcal{M}}$ *if* $X \in \mathcal{V}'_{\mathcal{F}}$ *and* $A \in \mathcal{F}^{\mathcal{M}}$

The purpose of formulas with meta-variables is double:

- Each quantification bound an arbitrary number of variables.
- They contain meta-variables which we will use as unknown formula. The arguments of a meta-variable correspond to values substituted to an above quantification. For example, if we want an unknown formula, we will take $\forall X \alpha^0(X)$.

We consider that the set $\mathcal{V}_{\mathcal{F}}$ used to define formulas was defined as $\mathcal{V}'_{\mathcal{F}} \times \mathbb{N}$ allow us to say that X^i is both a formula with meta-variable and a normal formula. This trick will simplify the following definitions (example: $\forall X(X^1 \to X^2)$ is a formula with meta-variable and $\forall X^1 \forall X^2 (X^1 \to X^2)$ is a normal formula).

Definition 5.2 (Variable-substitution) *We write* $A[X \Leftarrow Y]$ *for the formula* A *where* X *has been replaced by* Y*. This means that* X^i *is replaced by* Y^i *and* X *is replaced by* Y *when it is an argument of a meta-variable.*

Similarly we write $A[X \Leftarrow \alpha]$ *for the formula* A *where* X *has been replaced by* $\alpha \in \mathcal{M}_1$*. This means that* X^i *is replaced by* $\forall Y \alpha^i(Y)$ X *is replaced by* α *when it is an argument of a meta-variable.*

Example: $(X^i \to \beta^j(X))[X \Leftarrow \alpha] = (\forall Y \alpha^i(Y)) \to \beta^j(\alpha)$

The next definition are the atomic pieces to give values to meta-variables. We will distinguish four cases (cf. definition 5.6):

- Θ For an undefined meta-variable.
- Π For a meta-variable using one of its argument (similar to the projection case in the higher-order unification).
- Ξ, Ψ For a meta-variable whose value is respectively a variable or an arrow (similar to the imitation case in the higher-order unification).

Definition 5.3 (Meta-substitution) *A meta substitution* Σ *is a mapping from* $\bigcup_{n \in \mathbb{N}} (\mathcal{M}_n \times \mathbb{N})$ *to the set* $\{\Theta\} \cup \{\Pi(i,j); i,j \in \mathbb{N}\} \cup \{\Xi(X,i); X \in \mathcal{V}_{\mathcal{F}}, i \in \mathbb{N}\} \cup \{\Psi(\alpha,\beta); \alpha \in \mathcal{M}_n, \beta \in \mathcal{M}_p, n,p \in \mathbb{N}\}$*. Moreover, we must have the following condition:*

- $\Sigma(\alpha,i) = \Theta$ *for all* (α,i) *except a finite number of pairs.*
- *if* $\Sigma(\alpha,i) = \Pi(j,k)$ *and* $\alpha \in \mathcal{M}_n$ *then* $1 \leq k \leq n$
- *if* $\Sigma(\alpha,i) = \Psi(\beta,\gamma)$ *and* $\alpha \in \mathcal{M}_n$ *then* $\beta \in \mathcal{M}_{n+1}$ *and* $\gamma \in \mathcal{M}_n$

Definition 5.4 *We write* $\Sigma[(\alpha,i) \Leftarrow v]$ *for the meta-substitution* Σ' *defined by* $\Sigma'(\alpha,i) = v$ *and* $\Sigma'(\beta,j) = \Sigma(\beta,j)$ *if* $\alpha \neq \beta$ *or* $i \neq j$*.*

Definition 5.5 *We write* Σ_0 *for the substitution defined by* $\Sigma_0(\alpha,i) = \Theta$ *for all* α *and* i*.*

Definition 5.6 (Application of a meta-substitution) *If* Σ *is a meta-substitution and* $A = \alpha^i(\chi_1, \ldots, \chi_n) \in \mathcal{F}^{\mathcal{M}}$*, we define* $A\Sigma \in \mathcal{F}^{\mathcal{M}}$ *as follows:*

- *if* $\Sigma(\alpha,i) = \Theta$ *then* $A\Sigma = A$
- *if* $\Sigma(\alpha,i) = \Pi(j,k)$ *and* $\chi_j = X$ *then* $A\Sigma = X^k$
- *if* $\Sigma(\alpha,i) = \Pi(j,k)$ *and* $\chi_j = \beta$ *then* $A\Sigma = \forall X\, \beta^k(X)$
- *if* $\Sigma(\alpha,i) = \Xi(X,j)$ *then* $A\Sigma = X^j$
- *if* $\Sigma(\alpha,i) = \Psi(\beta,\gamma)$ *then* $A\Sigma = (\forall X \beta^0(X, \chi_1, \ldots, \chi_n)) \to \gamma^0(\chi_1, \ldots, \chi_n)$

Definition 5.7 (Full application of a meta-substitution) *If Σ is a meta-substitution and A is a meta-formula, we define $A^\Sigma \in \mathcal{F}$ as follows:*

- *If $A = B \to C$ then $A^\Sigma = B^\Sigma \to C^\Sigma$*
- *If $A = X^i$ then $A^\Sigma = X^i$*
- *If $A = \forall X\, A$ then $A^\Sigma = \forall X^{i_1} \ldots \forall X^{i_n}\, A^\Sigma$ where $X^{i_1}, \ldots, X^{i_n}$ are all the variables of the form X^i occurring in A and where $i_1 < i_2 < \ldots < i_n$.*
- *If $A = \alpha^i(\chi_1, \ldots, \chi_n)$ we distinguish the following cases:*
  - *if $\Sigma(\alpha, i) = \Theta$ then $A^\Sigma = \forall X^0\, X^0$*
  - *if $\Sigma(\alpha, i) \neq \Theta$ then $A^\Sigma = (A\Sigma)^\Sigma$*

The mapping $A \mapsto A^\Sigma$ is partial. Indeed, if there are loops in the substitution Σ. Example: If $\Sigma(\alpha, 0) = \Psi(\beta, \alpha)$ then $\alpha(X)\Sigma$ is undefined.

Definition 5.8 *We define $\mathcal{M}(A)$ the set of free meta-variables of A by*

- $\mathcal{M}(X) = \emptyset$
- $\mathcal{M}(A \to B) = \mathcal{M}(A) \cup \mathcal{M}(B)$
- $\mathcal{M}(\forall X\, A) = \mathcal{M}(A)$
- $\mathcal{M}(\alpha^i(l)) = \{(\alpha, i)\}$

Definition 5.9 (Gensym) *We assume a function $\Phi(E)$ generating "fresh" elements of the set E. More precisely, when using Φ, we will always be using a "state" S of the algorithm. $\Phi(E)$ will choose a variable not occurring in S. In fact we should write $\Phi(E, S)$.*

Definition 5.10 (The subtyping algorithm) *A state of the subtyping algorithm is a triplet $\Sigma; C \vdash I$ where*

- *I is a set of inequation of the form $A \subset_\Gamma B$ with $A, B \in \mathcal{F}^\mathcal{M}$ and Γ is a finite subset of $\mathcal{F}^\mathcal{M}$.*
- *S is a meta-substitution*
- *C is a set of constraints of the form (X, Γ) where $X \in \mathcal{V}'_\mathcal{F}$ and Γ is a finite subset of $\mathcal{F}^\mathcal{M}$.*

failure condition *If the state $\Sigma; C \vdash I$ fulfills one of the following conditions, then the algorithm fails:*

- *There exists $(X^i \subset_\Gamma Y^j) \in I$ with $X^i \neq Y^j$.*
- *There exists $(X^i \subset_\Gamma A \to B) \in I$.*
- *There exists $(A \to B \subset_\Gamma X^i) \in I$.*
- *There exists $(X, \Gamma) \in C$, $A \in \Gamma$ and $i \in \mathbb{N}$ such that X^i occurs free in A^Σ.*

steps *If we did not encounter a failure condition, we can apply one of the following steps (the initial state is the conclusion of the rule and the resulting state is the first premise of the rule, the other premises being conditions to apply this step):*

$$\frac{\Sigma; C \vdash I}{\Sigma; C \vdash X^i \subset_\Gamma X^i, I}\ \textit{Axiom}$$

$$\frac{\Sigma; C \vdash B \subset_{\Gamma,C} B', A' \subset_{\Gamma,C} A, I}{\Sigma; C \vdash A \to B \subset_\Gamma A' \to B', I}\ \textit{Arrow}$$

$$\frac{\Sigma; C \vdash A[X \Leftarrow \alpha] \subset_\Gamma B, I \quad \alpha = \Phi(\mathcal{M}_1)}{\Sigma; C \vdash \forall X\, A \subset_\Gamma B, I}\ \textit{Left forall}$$

$$\frac{\Sigma; C, (Y, (\Gamma, B)) \vdash A \subset_\Gamma B[X \Leftarrow Y], I \quad Y = \Phi(\mathcal{V}'_{\mathcal{F}})}{\Sigma; C \vdash A \subset_\Gamma \forall X\, B, I}\ \textit{Right forall}$$

$$\frac{\Sigma; C \vdash \alpha^i(\chi_1, \ldots, \chi_n)\Sigma \subset_\Gamma B, I \quad \Sigma(\alpha, i) \neq \Theta}{\Sigma; C \vdash \alpha^i(\chi_1, \ldots, \chi_n) \subset_\Gamma B, I}\ \textit{Left substitution}$$

$$\frac{\Sigma; C \vdash A \subset_\Gamma \beta^i(\chi_1, \ldots, \chi_n)\Sigma, I \quad \Sigma(\beta, i) \neq \Theta}{\Sigma; C \vdash A \subset_\Gamma \beta^i(\chi_1, \ldots, \chi_n), I}\ \textit{Right substitution}$$

$$\frac{\Sigma[(\alpha, i) \Leftarrow \Psi(\beta, \gamma)]; C \vdash A \to B \subset_\Gamma \alpha^i(l), I \quad \Sigma(\alpha, i) = \Theta, \beta = \Phi(\mathcal{M}_{n+1}), \gamma = \Phi(\mathcal{M}_n)}{\Sigma; C \vdash A \to B \subset_\Gamma \alpha^i(l), I}\ \textit{arrow immitation}$$

$$\frac{\Sigma[(\alpha, i) \Leftarrow \Xi(X, j)]; C \vdash X^j \subset_\Gamma \alpha^i(l), I \in I \quad \Sigma(\alpha, i) = \Theta, X \notin l}{\Sigma; C \vdash X^j \subset_\Gamma \alpha^i(l), I \in I}\ \textit{constant immitation}$$

$$\frac{\Sigma[(\alpha, i) \Leftarrow \Pi(j, k)]; C \vdash A \subset_\Gamma \alpha^i(l), I \quad A = B \to C \text{ or } A = X^k \quad \Sigma(\alpha, i) = \Theta}{\Sigma; C \vdash A \subset_\Gamma \alpha^i(l), I \in I}\ \textit{projection}$$

Non-determinism *This steps are non-deterministic: We have often to choose between the projection step and the arrow imitation or constant imitation steps. Moreover, the projection step cover an infinity of cases (we will bring this back to a finite number in the section 7).*

Termination *If the algorithm does not fail and reach a state where no steps are applicable, then we say that it terminates. In the final steps all the inequations are of the form $\alpha^i(l) \subset_\Gamma A$ with $\Sigma(\alpha, i) = \Theta$ (otherwise there would be an applicable step).*

If we reach a termination step $\Sigma'; C' \vdash I'$ starting from an initial state $\Sigma; C \vdash I$, we write $\Sigma; C \vdash I \Longrightarrow \Sigma'$. The non-determinism implies that Σ' is in general not unique (if it exists).

Definition 5.11 (Typing algorithm) *From a meta-substitution Σ and a sequent $\Gamma \vdash t : A$ using meta-formulas, we construct a state. We write $\Sigma, \Gamma \vdash t : A \Longrightarrow \Sigma'; C \vdash I$ to say that the state $\Sigma'; C \vdash I$ is constructed from Σ and the sequent $\Gamma \vdash t : A$. Here are the rule of the typing algorithm:*

$$\frac{}{\Sigma, \Gamma, x : A \vdash x : B \Longrightarrow \Sigma; \emptyset \vdash A \subset_{\Gamma,A} B}$$

$$\frac{\Sigma, \Gamma, x : A \vdash t : B \Longrightarrow S}{\Sigma, \Gamma \vdash \lambda x\, t : A \to B \Longrightarrow S}$$

$$\frac{\Sigma, \Gamma \vdash t : \forall X\, \alpha^0(X) \to A \Longrightarrow \Sigma'; C_1 \vdash I_1 \quad \Sigma', \Gamma \vdash u : \alpha^0(X) \Longrightarrow \Sigma''; C_2 \vdash I_2 \quad \alpha = \Phi(\mathcal{M}_1)}{\Sigma, \Gamma \vdash (t\, u) : B \Longrightarrow \Sigma''; C_1, C_2, (X, \Gamma) \vdash I_1, I_2}$$

$$\frac{\Sigma[(\alpha, i) \Leftarrow \Psi(\beta, \gamma)], \Gamma \vdash \lambda x\, t : \alpha^i(\chi_1, \ldots, \chi_n) \Longrightarrow \Sigma'; C \vdash I \quad \Sigma(\alpha, i) = \Theta, \beta = \Phi(\mathcal{M}_{n+1}), \gamma = \Phi(\mathcal{M}_n)}{\Sigma, \Gamma \vdash \lambda x\, t : \alpha^i(\chi_1, \ldots, \chi_n) \Longrightarrow \Sigma'; C \vdash I}$$

$$\frac{\Sigma, \Gamma \vdash t : \alpha^i(\chi_1, \ldots, \chi_n)\Sigma \Longrightarrow \Sigma'; C \vdash I \quad \Sigma(\alpha, i) \neq \Theta}{\Sigma, \Gamma \vdash t : \alpha^i(\chi_1, \ldots, \chi_n) \Longrightarrow \Sigma'; C \vdash I}$$

Failure This algorithm fails if no rules are applicable.

Definition 5.12 (The whole algorithm) *It consists in applying the typing algorithm to an initial sequent $\Gamma \vdash t : A$ and the identity meta-substitution Σ_0 to get a state. Then one applies the subtyping algorithm to this state to get a final substitution Σ. In this case, we write $\Gamma \vdash t : A \Longrightarrow \Sigma$ which means that $\Sigma_0; \Gamma \vdash t : A \Longrightarrow \Sigma'; C \vdash I$ and $\Sigma'; C \vdash I \Longrightarrow \Sigma$*

6 Correctness and Completeness

To prove the correctness, we need a few definitions:

Definition 6.1 *If $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, we write Γ^Σ for $x_1 : A_1^\Sigma, \ldots, x_n : A_n^\Sigma$*

Definition 6.2 *We say that Σ satisfies a set of inequations I if for all $A \subset_\Gamma B \in I$ we have $A^\Sigma \subset_{\Gamma^\Sigma} B^\Sigma$ derivable. We write $\Sigma \models I$.*

We say that Σ satisfies a set of constraints C if for all $(X, \Gamma) \in C$ and $i \in \mathbb{N}$ we have $X^i \notin \mathcal{V}(\Gamma^\Sigma)$. We write $\Sigma \models C$.

Definition 6.3 *We define the relation $<$ between meta-substitution by $\Sigma < \Sigma'$ if and only if $\Sigma(\alpha, i) \neq \Theta$ implies $\Sigma'(\alpha, i) = \Sigma(\alpha, i)$.*

Lemma 6.4 (Correctness of the subtyping algorithm) *If $\Sigma, C \vdash I \Longrightarrow \Sigma'$ then we have $\Sigma' \models C$, $\Sigma' \models I$ and $S < S'$.*

Proof: The proof is immediate by induction on the derivation of $\Sigma, C \vdash I \Longrightarrow \Sigma'$. The projection and imitation steps define the substitution Σ', respecting the third properties. The initial constraints set is always kept and the failure of the second properties implies the failure of the algorithm. All the rules modifying the set of inequations corresponds exactly to one of the rule of the subtyping system (the second properties giving the needed condition for the $\forall_r^\subset$ rule). ■

Theorem 6.5 (Correctness of the whole algorithm) *If $\Gamma \vdash t : A \Longrightarrow \Sigma$, then $\Gamma^\Sigma \vdash_s t : A^\Sigma$ is derivable in the simplified system.*

Proof: $\Gamma \vdash t : A \Longrightarrow \Sigma$, Means that $\Sigma_0, \Gamma \vdash t : A \Longrightarrow \Sigma', C \vdash I$ and $\Sigma', C \vdash I \Longrightarrow \Sigma$ with $\Sigma_0(\alpha, i) = \Theta$ for all α and i.

We prove the theorem by induction on the derivation of $\Sigma_0, x_1 : \Gamma \vdash t : A \Longrightarrow \Sigma', C \vdash I$. Three of the rules correspond exactly to the rule of the simplified system $\vdash_s$. The other two rules participate in the construction and application of Σ' and do not change the sequent after substitution by Σ. The lemma 6.4 is used for the axiom case and the condition on the variables in the implication elimination case. ■

For the completeness proof, we need to define the notion of complete substitution Σ for a set of meta-variables E: It means that each meta-variables α^i in E as a full value in Σ without using the Θ cases.

Definition 6.6 *We say that Σ is n-complete for a set $E \subset \bigcup_{n \in NN}(\mathcal{M}_n \times \mathbb{N})$ iff*

- *$n = 0$ and for all $(\alpha, i) \in E$, $\Sigma(\alpha, i) = \Pi(j, k)$ or $\Sigma(\alpha, i) = \Xi(X, j)$*
- *$n > 0$ and Σ is n-complete for $E \cup \{\beta, \exists(\alpha, i) \in E, \Sigma(\alpha, i) = \Phi(\beta, \gamma)\} \cup \{\gamma, \exists(\alpha, i) \in E, \Sigma(\alpha, i) = \Phi(\beta, \gamma)\} - \{\alpha, \exists(\alpha, i) \in E, \Sigma(\alpha, i) = \Phi(\beta, \gamma)\}$*

We say that Σ is complete for E if there exists $n \in \mathbb{N}$ such that Σ is n-complete for E.

It is clear that Σ complete for E and $\mathcal{M}(A) \subset E$ implies that $A^{\downarrow}S$ is defined (this is the purpose of this definition).

Lemma 6.7 *If $\Sigma < \Sigma_1$, if Σ_1 complete for E, if $(\alpha, i) \in E$, if $\Sigma_1(a, i) = \Psi(\beta', \gamma')$ and if β, γ are fresh (for all j, $(\beta, j) \notin E$, $(\gamma, j) \notin E$ and $\Sigma(\beta, j) = \Sigma(\gamma, j) = \Theta$) then there exists Σ_2 complete for E such that $\Sigma[(\alpha, i) \Leftarrow \Psi(\beta, \gamma)] \subset \Sigma_2$ and $A^{\Sigma_1} = A^{\Sigma_2}$ for any formula such that $\mathcal{M}(A) \subset E$.*

Proof: We first can assume that β', γ' are not used by Σ_1 (if it is not true, as $\beta', \gamma' \notin E$ we can replace them uniformly by two other variables in Σ_1). Then one can take for all $i \in \mathbb{N}$ $\Sigma_2(\beta, i) = \Sigma_1(\beta', i)$ and $\Sigma_2(\gamma, i) = \Sigma_1(\gamma', i)$. This implies all the wanted properties. ■

Lemma 6.8 *If $\Sigma < \Sigma_1$, if Σ_1 complete for E, if α is fresh (for all j, $(\alpha, j) \notin E$ and $\Sigma(\alpha, j) = \Theta$) then for any formula B, we can find Σ_2 such that $(\forall X\, \alpha^i(X))^{\Sigma_2} = B$, $\Sigma < \Sigma_2$ and $A^{\Sigma_1} = A^{\Sigma_2}$ for any formula such that $\mathcal{M}(A) \subset E$.*

Proof: Similar to the proof of the previous theorem. ■

Lemma 6.9 (Completeness of subtyping algorithm) *If $\Sigma' < \Sigma_1$, if Σ_1 complete for E, if $\Sigma_1 \models C$, if $\Sigma_1 \models I$ and if the free meta-variable of C and I are in E, then we $\Sigma'; C \vdash I \Longrightarrow \Sigma$ and we can find Σ_2 such that $\Sigma < \Sigma_2$ and $A^{\Sigma_1} = A^{\Sigma_2}$ for any formula such that $\mathcal{M}(A) \subset E$.*

Proof: We prove this result by induction on the sum of the size of all derivation in $\Sigma_1 \models I$. The proof uses exactly the same method as the proof of the next theorem (which is more detailed) using lemma 6.7 and 6.8. ■

Definition 6.10 *We say that a formula A is strict, if for all sub-formulas of A of the shape $\alpha^i(l)$, l do not use meta-variables. We say that a context is strict if each formula in the context is strict.*

Lemma 6.11 (Completeness of the typing algorithm) *If Γ, A are strict, if Σ_1 is complete for a set E, if $\mathcal{M}(\Gamma) \cup \mathcal{M}(A) \subset E$, if $\Gamma^{\Sigma_1} \vdash_s t : A^{\Sigma_1}$ and if $\Sigma' < \Sigma_1$, then there exists Σ_2 such that $\Sigma'; \Gamma \vdash t : A \Longrightarrow \Sigma; C \vdash I$, $\Sigma < \Sigma_2$, $\Sigma_2 \models C$, $\Sigma_2 \models I$ and $B^{\Sigma_2} = B^{\Sigma_1}$ for any formula such that $\mathcal{M}(B) \subset E$.*

Proof: We do an induction on the proof of $\Gamma^{\Sigma_1} \vdash_s t : A^{\Sigma_1}$:

- If the last rule is the axiom: Then $t = x$, $x : B^{\Sigma_1} \in \Gamma^{\Sigma_1}$ and $B^{\Sigma_1} \subset_{\Gamma^{\Sigma_1}} A^{\Sigma_1}$. The typing algorithm gives $\Sigma'; \Gamma \vdash t : A \Longrightarrow \Sigma'; \emptyset \vdash B \subset_{\Gamma} A$ which satisfies the wanted properties if we take $\Sigma_2 = \Sigma_1$.
- If the last rule is the implication elimination, we have $t = (u\, v)$, $\Gamma^{\Sigma_1} \vdash_s v : B$, $\Gamma^{\Sigma_1} \vdash_s u : (\forall X_1, \ldots, X_n\, B) \rightarrow A^{\Sigma_1}$, and $X_1, \ldots, X_n \notin \mathcal{V}(\Gamma^{\Sigma_1})$.

  When we apply the typing algorithm to $\Sigma'; \Gamma \vdash t : A$, we first must apply it to $\Sigma'; \Gamma \vdash u : (\forall X\, \alpha^0(X)) \rightarrow A$ where α is a fresh variable. Using lemma 6.8, we can extend Σ_1 into Σ_2 complete for $E \cup \{(\alpha, 0)\}$ to get $(\forall X\, \alpha^0(X))^{\Sigma_2} = \forall X_1, \ldots, X_n\, B$ and $D^{\Sigma_1} = D^{\Sigma_2}$ for any formula such that $\mathcal{M}(D) \subset E$.

By induction hypothesis (using the set $E \cup \{(\alpha, 0)\}$) we find Σ_3 such that $\Sigma'; \Gamma \vdash u : (\forall X\, \alpha^0(X)) \to A \Longrightarrow \Sigma''; C_1 \vdash I_1$, $\Sigma'' < \Sigma_3$, $\Sigma_3 \models C_1$, $\Sigma_3 \models I_1$ and $C^{\Sigma_3} = C^{\Sigma_2}$ for any formula such that $\mathcal{M}(C) \subset E$. Thus we have $\Gamma^{\Sigma_1} = \Gamma^{\Sigma_3}$ and $(\forall X_1, \ldots, X_n\, B) \to A^{\Sigma_1} = ((\forall X\, \alpha^0(X)) \to A)^{\Sigma_3}$,

Then we have to apply the typing algorithm to the state $\Sigma''; \Gamma \vdash v : \forall X\, \alpha^0(X)$. We still have $\Sigma'' < \Sigma_3$ and $\Gamma^{\Sigma_3} \vdash_s v : (\forall X\, \alpha^0(X))^{\Sigma_3}$. We can apply the induction hypothesis with the set $E' = E \cup \{(\alpha, 0)\} \cup \mathcal{M}(I_1) \cup \mathcal{M}(C_1)$ and we get Σ_4 such that $\Sigma''; \Gamma \vdash v : \forall X\, \alpha^0(X) \Longrightarrow \Sigma; C_2 \vdash I_2$, $\Sigma < \Sigma_4$, $\Sigma_4 \models C_2$, $\Sigma_4 \models I_2$ and $C^{\Sigma_4} = C^{\Sigma_3}$ for any formula such that $\mathcal{M}(C) \subset E'$.

Thus we have $\Sigma''; \Gamma \vdash t : A \Longrightarrow \Sigma; C_1, C_2, (X, \Gamma) \vdash I_1, I_2$ and Σ_4 is the wanted substitution because we have $\Sigma < \Sigma_4$, $\Sigma_4 \models C_2$, $\Sigma_4 \models C_1$ (because $\mathcal{M}(C_1) \subset E'$), $\Sigma_4 \models I_2$, $\Sigma_4 \models I_1$ (because $\mathcal{M}(I_1) \subset E'$), and $X \notin \Gamma^{\Sigma_4} = \Gamma^{\Sigma_1}$

- If the last rule is the implication introduction and if $A = B \to C$ the proof is similar (but much more simple) to the previous case.
- If the last rule is the implication introduction, if $A = \alpha^i(l)$ and if $\Sigma'(a, i) \neq theta$. As the formula A is strict we must have $\Sigma_1(\alpha, i) = \Psi(\beta, \gamma)$. As $\Sigma' < \Sigma_1$ we have also $\Sigma'(\alpha, i) = \Psi(\beta, \gamma)$. Therefore, applying the typing algorithm to the state $\Sigma'; \Gamma \vdash t : A$ gives the same result as applying it to $\Sigma'; \Gamma \vdash t : \forall X\, \beta^0(X, l) \to \gamma^0(l)$ which was treated in the previous case using $\forall X\, \beta^0(X, l) \to \gamma^0(l)$ in place of A.
- If the last rule is the implication introduction, if $A = \alpha^i(l)$ and if $\Sigma'(a, i) = theta$. As in the previous case we have $\Sigma_1(\alpha, i) = \Psi(\beta, \gamma)$. Therefore, applying the typing algorithm to the state $\Sigma'; \Gamma \vdash t : A$ gives the same result as applying it to $\Sigma'[(a, i) \Leftarrow \Psi(\beta', \gamma')]; \Gamma \vdash t : A$. Then we can extend Σ_1 using the lemma 6.7 to get a substitution Σ_2 still complete for E and such that $D^{\Sigma_2} = D^{\Sigma_1}$ if $\mathcal{M}(D) \subset E$. Then we can conclude as in the previous case using Σ_2 in place of Σ_1. ■

Theorem 6.12 (Completeness of the whole algorithm) *If Γ, A are strict, if Σ_1 is complete for $\mathcal{M}(\Gamma) \cup \mathcal{M}(A)$ and if $\Gamma^{\Sigma_1} \vdash_s t : A^{\Sigma_1}$ then there exists a substitution Σ such that $\Gamma \vdash t : A \Longrightarrow \Sigma$.*

Proof: This is just the composition of the two previous lemma. ■

Corollary 6.13 (Type-checking and Type-inference) *Using the completeness and correctness of the simplified system and of our algorithm, we can give the following type-checking and type-inference procedure for the system F^η:*

To check that $\Gamma \vdash t : A$, First you take A'' starting with no quantifiers such that $\pi(A) = \forall X_1, \ldots, X_n\, A''$. Then you need to build a formula A' and a context Γ' with meta-variables (which actually will not use meta-variables!) such that $A = A'^{\Sigma_0}$ and $\Gamma = \Gamma'^{\Sigma_0}$. Then you can use the complete algorithm on $\Gamma' \vdash t : A'$ which will terminates if and only if $\Gamma \vdash t : A$ is derivable.

For inferring a type for t in a context Γ, construct Γ' as above and use the complete algorithm on $\Gamma' \vdash t : \forall X\, \alpha^0(X)$ where α is a fresh meta-variable. This algorithm stops on a substitution Σ if and only if the exists Σ' such that $\Gamma \vdash t : (\forall X\, \alpha^0(X))^{\Sigma'}$.

Proof: Immediate consequence of the theorem 6.12, 6.5, 4.6 and 4.1. ■

7 Implementation and optimization

Finitely branching search The projection rule contains an infinity of cases. This can easily be corrected: When we need to apply the projection rule, we have an inequation of the shape $A \subset_\Gamma \alpha^i(\chi_1, \ldots, \chi_l)$. For all i, χ_i relates to formulas which have been introduced by eliminating a quantification in a formula $\forall X\, B$. If we keep track of this original quantification [1], we can know the variables $X^1, \ldots, X^n$ which have already been used in B and we can restrict the projection for i to $\Pi(i, 1), \ldots \Pi(i, n+1)$: either we use an already used variable or a new one.

[1] this is not difficult, but we have omitted this to try to make the algorithm as readable as possible

Using the commutation of rules If is clear that we can always delay the imitation and the projection rules. This is important: no need to try to instantiate a meta-variable if the algorithm will fail anyway using other rules.

Clever failure When one branch of the search fails, this means we have to backtrack to search another branch. When backtracking, we will try another value for some meta-variable. It is important not to try other values if the previous value of the meta variable was not part of the failure. It is easy to keep track of this by storing for each inequation the set of meta-variables whose values have been used to get it (these are the meta-variables used by the substitution rules to leads to this inequation). We must also keep such a set for the constraints.

Bounding the search To search a tree which may have infinite branch we need to bound the search. We have found a bound which make the algorithm very efficient. We first need to give the following definition:

Definition 7.1 *If $S = \Sigma, \mathrm{X} \vdash I$ is a state of the subtyping algorithm, we define two relations $=_S$ and $<_S$ on $\bigcup_{n\in\mathbb{N}}(\mathcal{M}_n \times \mathbb{N})$ as the smallest relations such that:*

- *$=_S$ is an equivalence relation*
- *$<_S$ is transitive and compatible with $=_S$*
- *$(a,i) <_S (b,j)$ if $\Sigma(\alpha,i) = \Psi(\beta,\gamma)$ or $\Sigma(\alpha,i) = \Psi(\gamma,\beta)$*
- *$(a,i) =_S (b,j)$ if $\alpha^i(l) \subset \beta^j(l')$ in I.*
- *$(a,i) <_S (b,j)$ if B does not start with a meta-variable, if $\alpha^i(l) \subset_\Gamma B$ or $B \subset_\Gamma \alpha^i(l)$ in I and if $(b,j) \in \mathcal{M}(B)$.*

This relation could seem difficult to compute, but in fact it can be computed by associating a rational first order unification problem to the state and solving it (in pseudo linear time).

We say that an arrow imitation "cost" is the variable (a,i) that will be instantiated by this rule is part of a cycle in the relation $<_R$. Then, we can bound the search by limiting the number of arrow imitation that "cost". Unfortunately we are not yet able to prove that this limitation really makes all the branch finite[2]. However all the experiments have fail to make our algorithm loop with this bound. So we conjecture that our bounding method is correct.

8 Experimentation

We have implemented the present algorithm and tested it on various program on Church numerals and lists. On small programs (such as various predecessors or storage operators) the algorithm always succeed very fast. Nevertheless, some of these programs use the polymorphism in non trivial way.

We have also try one bigger program: David's Inf [2]. It is a Term computing the "less or equal" test on two Church numerals n and m in time $o(\inf(n\ln(n), m\ln(m))$. This program is medium size (an untyped λ-term of around 50 lines) and uses a lot of polymorphism (it contains a storage operator and a predecessor on binary numeral represented as list of booleans). Our algorithm fails to type this term in a reasonable time (it reaches the "cost" 32 is a few days before we stopped it). However if we give the type of the important subterms, the all program is typed in a few seconds.

From this experiments, we can hope that this algorithm may be useful in a real programming language including ML-like data-types. Indeed, in such a language, programs contains much more implicit typing information and there is a good chance that all useful programs could be typed in a feasible time if the programmer adds a reasonable amount of type information in the program (probably not more that what is needed to get a readable program).

[2] we can prove it if we also associate a cost to the projection rule. But this make the algorithm more inefficient specially when we add more type information which is the opposite of the desired property

9 Further Work

The main work that remains is to prove (or disprove) that our bounding method is correct. On this subject, it would also be interesting to know if this bounding method corresponds to a stratification of the system that could be defined in a nice way. We already know that the level 0 of such stratification will strictly contain the simply-typed λ-terms (there are polymorphic terms which are type-checkable using no arrow imitation rule that "cost": for instance $(\lambda x\,(x\ x))\lambda y\,y)$.

We also know how to extend our algorithm to recursive type and ML-like data-types. We need to implement this to further check the practical feasibility of our algorithm.

References

[1] L. Damas and R. Milner. Principal type schemes for functional programs. In *Ninth Annual ACM Symposium on the Principles of Programming Languages*, pages 207–212. ACM, 1982.

[2] R. David. The inf function if type system f. *Theoretical Computer Science*, 135:423–431, 1994.

[3] P. Giannini and S. Ronchi Dela Rocca. A type inference algorithm for a stratified polymorphic type discipline. In *Information and Computation*, pages 115–173, 1994.

[4] J.-Y. Girard. *Interprétation Fonctionelle et Élimination des Coupures de l'Arithmétique d'Ordre Supèrieur*. PhD thesis, University Paris VII, 1972.

[5] J.-Y. Girard. The system F of variable types: fifteen years later. *Theoretical Computer Science*, 45:159–192, 1986.

[6] G. Huet. A unification algorithm for typed λ-calculus. *Theoritical Computer Science*, 1:27–57, 1975.

[7] A. J. Kfoury and J. Tiuryn. Type reconstruction in finite-rank fragments of the second order λ-calculus. *Inf. Comput.*, pages 228–257, 1992.

[8] A. J. Kfoury and J. B. Wells. A direct algorithm for type inference in the rank-2 fragments of the second-order λ-calculus. In *LIPS*, pages 176–185. ACM, 1994.

[9] G. Longo, K. Milsted, and S. Soloviev. A logic of subtyping. In *Logic in Computer Science*, 1995.

[10] J.C. Mitchell. Polymorphic type inference and containment. *Information and Computation*, 76:211–249, 1988.

[11] Kanellakis P.C. and Mitchell J.C. Polymorphic unification of ml typing. Technical report, Brown University, 1989.

[12] Piperno and Ronchi della Rocca. Type inference and extensionality. In *Logic in Computer Science*, pages 196 – 205. IEEE Computer Society Press, 1994.

[13] J. C. Reynolds. Towards a theory of type structure. In *Symposium on Programmings*, volume 19, pages 408–425. Springer Verlag, 1974. Lecture Notes in Computer Science.

[14] J. Tiuryn and P. Urzyczyn. The subtyping problem for second-order types is undecidable. In *Logic in Computer Science*, 1996.

[15] J. B. Wells. Typability and type checking in the second order λ-calculus are equivalent and undecidable. In *Logic in Computer Sciences*, pages 176–185. IEEE, 1994.