TYPE CHECKING IN SYSTEM $\mathbf{F}^\eta$

Christophe Raffalli
email: raffalli@logique.jussieu.fr

Abstract

The main contribution of this paper is a partial type-checking algorithm for the system $\mathbf{F}^\eta$ and its use in a programming language like ML. We define this system as an extension of the second-order λ-calculus (system $\mathbf{F}$) verifying the preservation of type during computation (subject-reduction) for $\beta\eta$-reduction (this result fails for η-reduction in system $\mathbf{F}$). Our presentation is based on an original notion of sub-typing which includes all the handling of quantification rules.

1 Introduction.

Motivation. Type systems have proved to be useful for many modern functional programming languages such as ML, Miranda, Haskell, ... In most cases, the basis of the type system is Milner's algorithm [12]. The main characteristic of these type systems is polymorphism which allows the programmer to write generic functions that can work on arguments of different types. However it is often insufficient: polymorphic recursion, existential types or the state monad of Haskell are treated using specific extensions; construction of references on polymorphic objects are illegal. All these examples could be typed in a uniform way using second-order λ-calculus (called system $\mathbf{F}$), which was formulated independently by Girard [4] and Reynolds [15]. There is no functional language directly using system $\mathbf{F}$, because both problems of type-checking (verifying if a given term has a given type) and type-inference (construction of a type for a given term) were open for a long time, and are now known to be undecidable [18].

The goal of this work is to define an extension of system $\mathbf{F}$ for which a typing algorithm can be shown to be useful as a type system. Therefore we shall state the properties we believe to be desirable for such an algorithm:

- It should allow some type inference: one does not want to write the type for all functions.
- It must be able to deliver error messages that are helpful to correct the program.
- It should be able to use some type information provided by the programmer. Indeed an error can be detected very late because it resulted only in an incorrect type being inferred for a previously defined function. To give some type information and rerun the type-system is one of the best ways to localize the error.
- It should be feasible. This does not necessarily mean that the typing must always be efficient or terminating. It is enough when interrupted to give a clear message that help the programmer to locate the parts of the code that are difficult to type. However, the user should always be able to add enough typing information to get a reasonable computing time.
- It should be as simple as possible. To go from pure λ-calculus to a real programming language many features need to be added. This requires extensions to the type system and this is greatly simplified if the basic type system for λ-calculus is simple.

Contributions of this paper. The initial motivation of this work is the problem of type preservation (subject-reduction) during η-reduction for system **F** in Curry's style. Indeed a term which η-reduces to the identity function can be given a type which is not admissible for the identity function itself. The system $\mathbf{F}^\eta$ introduced in this paper fixes this problem. This system is therefore equivalent to system **F** with the η-reduction rule and so it is also equivalent to Mitchell system [13] (see the section on related work for a more detailed comparison). The system $\mathbf{F}^\eta$ uses a new notion of sub-typing to deal with the quantifier rules. It allows more freedom in the manipulation of quantifiers. One of the consequences is that types are more general. For instance $\forall X(X \to X)$ is a principal type for the identity function (which means that $\forall X(X \to X)$ is a subtype of any other type for the identity function). This is not true for system **F**. However, there are still many terms which have no principal type in system $\mathbf{F}^\eta$. Section 3 introduces the definition and the results about this new type system. We prove strong normalization of typed terms (theorem 3.9), subject-reduction for $\beta\eta$-reduction (theorem 3.15) and we prove that a term is typeable in system $\mathbf{F}^\eta$ if and only if it has an η-expansion typeable in system **F**(theorem 3.16).

After developing system $\mathbf{F}^\eta$, we study the question of its applications. We develop in section 4 a partial algorithm which, we believe could be used for a programming language, because it enjoys the following properties:

- It is correct (theorem 4.19)
- It is not complete but it is always possible to add enough type information in a term to make it typeable by the algorithm (theorem 4.20). Moreover the kind of type information one need to add does not require to understand the type system (one only need to give the type of some sub-terms). This is quite different from writing a term in Church's style to show that it is typeable in system **F**.
- It never backtracks. Therefore understandable error messages can be generated to help localizing the problem.
- It stays simple and it is an extension of Milner's algorithm [12]. Milner's algorithm is a special case of our algorithm when it is applied to quantifier free terms. The sub-typing algorithm is then simplified into a unification algorithm (see lemma 4.16 and theorem 4.21). This implies that any language using Milner's algorithm can be extended to use our algorithm keeping backward compatibility with the initial language.
- The algorithm does not always terminate. However, it is unlikely to encounter a loop in a real program. Moreover, when the user interrupts the type system, a message can be printed to help the user to add type annotations to simplify the search.
- There is no bound to the amount of type information needed for type-checking (this would imply the decidability of the type-checking). However, experiments with a first implementation (reported in section 4.5) showed that very little type information (if any) is needed for many terms typeable in system **F**. We still do not know of any useful programs requiring too much type-information.

Related work. A lot of work has been devoted to type systems and particularly to system **F**. The undecidability of both type-checking and type inference has been establish by Wells [18]. The decidability problem of many stratifications has been studied [3, 6, 7, 9, 10, 11]. It is our opinion that the algorithms developed so far are not as simple as the one we present. However, they are usually complete for well defined fragments of system **F**. On the other hand our algorithm works for the complete system $\mathbf{F}^\eta$(and therefore system **F** which is a sub-system)

but is neither complete nor terminating. It seems that these problems have little consequences for the practical use of our algorithm. Both the theoretical results and the experiments give hope that our system could by implemented in a full scale programming language[1].

The work by Mitchell [13] on system $\mathbf{F}$ with containment presents another system using sub-typing which is equivalent to our presentation of system $\mathbf{F}^\eta$. The main difference is that Mitchells system is much more non-deterministic. It is always possible to use the transitivity rule (which is not present in our system) and there are two rules to introduce a quantification on the left of the sub-typing relation. In our formulation, there is only one possible rule applicable (the left and right rule for quantification permutte, so it does not matter if one apply one after or before the other). The only source of non-determinism in our presentation is the fact that an arbitrarely formula is introduced by the left rule for quantification (which replace the elimination rule for the quantifier). This allows us to use technics similar to the first-order unification to guess in some cases the right formula.

Our work is also related to works on type-checking in $\mathbf{F}_<$ and $\mathbf{F}^\omega_<$ [1, 2, 16]. In these papers the type system is viewed in "Church's style" with type information as part of the syntax of terms. However, one can often let some of the type annotations unknown and let a first order unification algorithm guess their value. The main difference with our approach is the rigidity of the structure of types which implies that types are less general and that one needs to write more type annotations (for instance, permutation of quantifiers corresponds to a small function, while this is transparent in our type-system).

One should also note that the recent work by Tiuryn and Urzyczyn [17] establish that system $\mathbf{F}^\eta$ is undecidable. One should also cite the work by Piperno and Ronchi della Rocca [14] where another partial algorithm related to the η-expansion is presented. This algorithm is quite different from ours (It uses a limitation on the shape of the typing derivation and is not a direct extention of Milner's algorithm).

We believe that our presentation of system $\mathbf{F}^\eta$ is original because it is much more deterministic that the previous approach. This is this determinism that allows us to describe a partial algorithm which is much simpler that all the algorithm we know and which seems to be praticaly usefull.

2 Notation and preliminaries on System F.

We consider only "Curry style" system $\mathbf{F}$ 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 $(\mathtt{t}\ \mathtt{t_1}\ \ldots\ \mathtt{t_n})$ and left-associative: $(\mathtt{t}\ \mathtt{u}\ \mathtt{v})$ means $((\mathtt{t}\ \mathtt{u})\ \mathtt{v})$. Abstractions are written $\lambda\mathtt{x}\,\mathtt{t}$ with the largest possible scope for the binder: $(\lambda\mathtt{x}\,\mathtt{t}\ \mathtt{u})$ means $(\lambda\mathtt{x}\,(\mathtt{t}\ \mathtt{u}))$. 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 $F_1 \to \ldots \to F_{n-1} \to F_n$ and right-associative: $F \to G \to H$ means $F \to (G \to H)$. 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 $A_1, \ldots, A_n$. If E is a set of formulas, we write $\mathcal{V}(E)$ for the union of all $\mathcal{V}(A)$ for $A \in E$.

We write $[\chi_1, \ldots, \chi_n \Leftarrow \varphi_1, \ldots, \varphi_n]$ for the substitution of the variables χ_i by φ_i and ϵ for the constant substitution (for all $X \in \mathcal{V}$, $X\epsilon = X$). 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.

[1] If any one involved in the development of such a language is interested to try an experiment, please contact me.

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

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

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

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

$$\frac{A \in \mathcal{F}}{A \subset_\emptyset A}\, Ax^{\subset} \qquad \frac{B \subset_\gamma D \qquad C \subset_{\gamma'} A \qquad \mathcal{V}(C) \cap (\gamma \cup \gamma') = \emptyset}{A \to B \subset_\gamma C \to D}\, \to^{\subset}$$

$$\frac{A[X \Leftarrow C] \subset_\gamma B}{\forall X\, A \subset_\gamma B}\, \forall_l^{\subset} \qquad \frac{A \subset_\gamma B}{A \subset_{\gamma \cup \{X\}} \forall X\, B}\, \forall_r^{\subset}$$

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

We write $\mathcal{V}(\sigma)$ for the set of all $X \in \mathcal{V}_\mathcal{F}$ such that there exists $Y \in \mathcal{V}_\mathcal{F}$, $Y\sigma \neq Y$ and $X \in \mathcal{V}(Y\sigma)$. From this we have for all formula A and all substitution σ, $\mathcal{V}(A\sigma) \subset \mathcal{V}(A) \cup \mathcal{V}(\sigma)$. Moreover, we only use substitution with "finite support". That means that for all substitution σ, $X\sigma = X$ except for a finite number of variables. This implies that we can always chose a variable X such that $X\sigma = X$ and $X \notin \mathcal{V}(\sigma)$.

Finally, we use the notation $\mathcal{P}(E)$ for the set of all subsets of E and $E \to F$ for the set of all mappings from E to F. We also use the set $\mathcal{P}^f(E)$ of all finite subsets of E and the set $E \to_f F$ of finite mappings from E to F ($f \in E \to_f F$ if $f \in E \to F$ and $f(x) \neq x$ only for a finite number of $x \in E$). With this notation, the set of all substitutions is isomorphic to $\mathcal{V} \to_f \mathcal{F}$.

The rules of system **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. To give a simple counter example, just consider two formulas F, G. Then we can prove $\vdash \lambda \mathtt{x} \lambda \mathtt{y}\, (\mathtt{x}\, \mathtt{y}) : \forall X (F \to G) \to \forall X\, F \to \forall X\, G$. This term can be η-reduced to $\lambda x\, x$, but there is no way to reduce the proof.

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

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

The relation $\subset_\gamma$ on formulas, where γ is a set of variables called the set of constraints, is defined as the smallest relation derivable using the deduction rules given in table 2.

The meaning of this relation is the following: If we have $A \subset_\gamma B$ with $\{X_1, \ldots, X_n\} = \gamma$, 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. The set of constraints γ is needed because we want the relation $\subset$ to deal with the introduction of the quantification. The use of this set of constraint is the main difference between Mitchel's sub-typing rules and ours. One should also not that our sub-typing is not transitive (the lemma 3.6 gives a weak version of the transitivity) but deal both with the introduction and the elimination of the quantifier (While in Mitchell's system, the sub-typing only deals with the elimination of the quantifier).

Properties of the sub-typing. We have choosen to prove directly that our presentation of the sytem $\mathbf{F}^\eta$ 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 (in fact it seems that the same fundamental lemmas are needed). Moreover the direct proof shows how the derivation are transformed during the process of $\beta\eta$-reduction.

Lemma 3.1 *If $\Gamma \vdash_F \mathtt{t} : A$, $A \subset_\gamma B$ and $\mathcal{V}(\Gamma) \cap \gamma = \emptyset$ then we can find $\mathtt{t}'$ an η-expansion of $\mathtt{t}$ ($t' \triangleright_\eta t$) such that $\Gamma \vdash_F \mathtt{t}' : B$.*

Proof: The proof is by induction on the derivation of $A \subset_\gamma B$. The only non obvious case is the $\rightarrow^\subset$ rule. In this case, we assume $\Gamma \vdash_F \mathtt{t} : A$, $A = A' \rightarrow A''$, $B = B' \rightarrow B''$, $B' \subset_{\gamma'} A'$, $A'' \subset_\gamma B''$ and $\mathcal{V}(B') \cap (\gamma \cup \gamma') = \emptyset$. We choose a λ-variable $\mathtt{x}$ not free in $\mathtt{t}$, and we get $\Gamma' \vdash_F \mathtt{t} : A$ with $\Gamma' = \Gamma, \mathtt{x} : B'$ (weakening is admissible in system $\mathbf{F}$). The induction hypothesis implies $\Gamma' \vdash_F \mathtt{u} : A'$ with $\mathtt{u} \triangleright_\eta \mathtt{x}$ and then $\Gamma \vdash_F \lambda \mathtt{x}\, \mathtt{v} : B$ with $\mathtt{v} \triangleright_\eta (\mathtt{t}\, \mathtt{u})$. This implies $\lambda \mathtt{x}\, \mathtt{v} \triangleright_\eta \mathtt{t}$.■

Lemma 3.2 *If $A \subset_\gamma B$ then $\mathcal{V}(B) \cap \gamma = \emptyset$.*

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

Corollary 3.3 *In the $\forall_r^\subset$ rule, we can always assume $X \notin \gamma$*

Proof: If $X \notin \mathcal{V}(B)$ then X can be chosen arbitrarely to have $X \notin \gamma$. Otherwise lemma 3.2 gives the expected result. ■

Lemma 3.4 *If $A \subset_\gamma B$ and for all variables $X \in \gamma$, $X\sigma = X$ and $X \notin \mathcal{V}(\sigma)$ then $A\sigma \subset_\gamma B\sigma$.*

Lemma 3.5 *If $A \subset_\gamma B$ with $\{X_1, \ldots, X_n\} = \gamma$ (we assume $X_1, \ldots, X_n$ to be distinct) then for all distinct fresh variables $Y_1, \ldots, Y_n$ (variables with no occurrences in A, B, γ and any other finite set we like) we have $A[X_1, \ldots, X_n \Leftarrow Y_1, \ldots, Y_n] \subset_{\{Y_1, \ldots, Y_n\}} B$.*

In the particular case where $\mathcal{V}(A) \cap \gamma = \emptyset$ this lemma tells us that we can change γ to contains only fresh variables. We call this particular case lemma 3.5i

It is important to note that when we apply any of the two previous lemmas the structure of the proof is left unchanged (rules order and proof size are not modified).

Proof: The proof of the two previous lemmas is done by simultaneous induction on the derivation of $A \subset_\gamma B$ and uses corollary 3.3. We start by exmining the case of lemma 3.4. We assume that A, B and σ fulfill the hypothesis of the lemma and we distinguish the following cases:

- If the last rule is the $Ax^\subset$ rule then the result is obvious.
- If the last rule is the $\forall_l^\subset$ rule. We have $A = \forall X\, A'$ and we deduced $A \subset_\gamma B$ from $A'[X \Leftarrow C] \subset_\gamma B$. As X is bound, we can chose it fresh (such that $X\sigma = X$ and $X \notin \mathcal{V}(\sigma)$). Then we can apply the induction hypothesis to find $A'[X \Leftarrow C]\sigma \subset_\gamma B\sigma$ which gives $A'\sigma[X \Leftarrow C\sigma] \subset_\gamma B\sigma$. Using the $\forall_l^\subset$ rule, we get $\forall X\, (A'\sigma) \subset_\gamma B\sigma$. And we get the expected result because $\forall X\, (A'\sigma) = (\forall X\, A')\sigma$.

- If the last rule is the $\forall_r^{\subset}$ rule, we have $B = \forall X\, B'$, $\gamma = \gamma' \cup \{X\}$ and we deduced $A \subset_\gamma B$ from $A \subset_{\gamma'} B'$. Then the induction hypothesis gives $A\sigma \subset_{\gamma'} B'\sigma$ (we can apply it because $\gamma' \subset \gamma$ and so σ fufills the needed hypothesis). Then we can apply the $\forall_r^{\subset}$ rule to find $A\sigma \subset_\gamma \forall X\, (B'\sigma)$ which gives the expected result because we get $\forall X\, (B'\sigma) = (\forall X\, B')\sigma$ from the hypothesis on σ and $X \in \gamma$.

- If the last rule is the $\rightarrow^{\subset}$ rule, we have $A = A' \rightarrow A''$, $B = B' \rightarrow B''$ and we have deduced $A \subset_\gamma B$ from $B' \subset_{\gamma'} A'$, $A'' \subset_\gamma B''$ and $\mathcal{V}(B') \cap (\gamma \cup \gamma') = \emptyset$. Then we can use lemma 3.5i and assume that γ' contains fresh variables such that σ fulfill the hypothesis of the lemma 3.4 for γ' too. This does not change the structure of the proof. Thus we can apply the induction hypothesis and find $A''\sigma \subset_\gamma B''\sigma$ and $A'\sigma \subset_{\gamma'} B'\sigma$. From the hypothesis on σ and the fresh choice of γ', we get $\mathcal{V}(B'\sigma) \cap (\gamma \cup \gamma') = \emptyset$. Thus we can apply the $\rightarrow^{\subset}$ rule to find the wanted result.

We can now consider the lemme 3.5. We assume that A, B and $X_1, \ldots, X_n, Y_1, \ldots, Y_n$ fulfill the hypothesis of the lemma. Moreover we will write σ for $[X_1, \ldots, X_n \Leftarrow Y_1, \ldots, Y_n]$ and γ_0 for $\{Y_1, \ldots, Y_n\}$. We distinguish the following cases:

- If the last rule is the $Ax^{\subset}$ rule then the result is trivial because $\gamma = \emptyset$.

- If the last rule is the $\forall_l^{\subset}$ rule, we have $A = \forall X\, A'$ and we deduced $A \subset_\gamma B$ from $A'[X \Leftarrow C] \subset_\gamma B$. The induction hypothesis gives $A'[X \Leftarrow C]\sigma \subset_{\gamma_0} B$. Thus we have $A'[X \Leftarrow C]\sigma = A'\sigma[X \Leftarrow C\sigma]$ (because we can always chose X such that $X \notin \gamma \cup \gamma_0$). So we can apply the $\forall_l^{\subset}$ rule to get the wanted result because the choice of X gives $\forall X\, (A'\sigma) = (\forall X\, A')\sigma$.

- If the last rule is the $\forall_r^{\subset}$ rule, we have $B = \forall X\, B'$, $\gamma = \gamma' \cup \{X\}$ and we deduced $A \subset_\gamma B$ from $A \subset_{\gamma'} B'$. Thus there exists i such that $X = X_i$. Moreover, using corollary 3.3, we can assume $X_i \notin \gamma'$. Thus we can apply lemmas 3.4 with $\sigma' = [X_i \Leftarrow Y_i]$ and we find $A\sigma' \subset_{\gamma'} B'\sigma'$. Now we can apply the induction hypothesis (because the size of the proof is unchanged) with $X_1, \ldots, X_{i-1}, X_{i+1}, \ldots, X_n$ to find $A\sigma \subset_{\gamma_0'} B'\sigma'$ with $\gamma_0' = \{Y_1, \ldots, Y_{i-1}, Y_{i+1}, \ldots, Y_n\}$. Then the $\forall_r^{\subset}$ rule gives $A\sigma \subset_{\gamma_0} \forall Y_i\, B'\sigma'$ which is the wanted result because $\forall Y_i\, B'\sigma' = \forall X\, B$.

- If the last rule is the $\rightarrow^{\subset}$ rule, we have $A = A' \rightarrow A''$, $B = B' \rightarrow B''$ and we have deduced $A \subset_\gamma B$ from $B' \subset_{\gamma'} A'$, $A'' \subset_\gamma B''$ and $\mathcal{V}(B') \cap (\gamma \cup \gamma') = \emptyset$. So we can apply the induction hypothesis and find $A''\sigma \subset_{\gamma_0} B''$. Moreover by induction hypothesis, with the particular case of lemma 3.5i, we can assume γ' to contains only fresh variables (not free in B', γ or γ_0 and such that σ fulfills the hypothesis of the lemma 3.4). Then we can apply the lemma 3.4 to find $B'\sigma \subset_{\gamma'} A'\sigma$, which implies $B' \subset_{\gamma'} A'\sigma$ because $\mathcal{V}(B') \cap (\gamma \cup \gamma') = \emptyset$ implies $B'\sigma = B'$. Moreover by hypothesis on γ_0, we have $\mathcal{V}(B') \cap (\gamma_0 \cup \gamma') = \emptyset$, thus we get the wanted result using the $\rightarrow^{\subset}$ rule on $A''\sigma \subset_{\gamma_0} B''$ and $B' \subset_{\gamma'} A'\sigma$. ■

Lemma 3.6 *If $A \subset_{\gamma_1} B$ and $B \subset_{\gamma_2} C$ then there exists a substitution σ and a set of variables γ_0 such that $A\sigma \subset_{\gamma_0} C$, and for all variable $X \notin \gamma_1$, $X\sigma = X$. We also can chose γ_0 such that $\gamma_0 - \gamma_2$ contains only fresh variables (not in any finite set we like). Moreover the number of rules used by the resulting proof is strictly smaller to the sum of the number of rules used by the two original proofs.*

Proof: It is an induction on the sum of the size of both proofs using lemmas 3.2 to 3.5. We can distinguish the following cases:

- If the last rule for any of the hypothesis is an axiom then the result follows easily from the induction hypothesis and lemma 3.5. In this case one rule disappear, so the total number of rules decreases by one.

- If the last rule in the proof of $A \subset_{\gamma_1} B$ is the $\forall_l^{\subset}$ rule or if the last rule in the proof of $B \subset_{\gamma_2} C$ is the $\forall_r^{\subset}$, then the result follows easily from the induction hypothesis. In this case we add one rule to replace another rule. So using the induction hypothesis, the number of rule decreases.

  If non of the previous cases apply only two possibilities remain. Indeed, the last rule for the first (resp. second) hypothesis can only be the $\forall_r^{\subset}$ (resp. $\forall_l^{\subset}$) or $\rightarrow^{\subset}$ rule. But these rules forces B to be either an arrow of a quantification. So here are the two last cases:

- The last rule for both hypothesis is the $\rightarrow^{\subset}$ rule. Without any lost of generality, we can assume that $\gamma_1 \cap \mathcal{V}(C) = \emptyset$ because we can apply lemma 3.5 to find a first substitution we can apply to A.

  Then, we have $A = A' \rightarrow A''$, $B = B' \rightarrow B''$ and $C = C' \rightarrow C''$ and we have deduced the hypothesis from

  $$C' \subset_{\gamma_2'} B' \subset_{\gamma_1'} A' \quad \text{and} \quad A'' \subset_{\gamma_1} B'' \subset_{\gamma_2} C''.$$

  with the extra conditions that $\mathcal{V}(B') \cap (\gamma_1 \cup \gamma_1') = \emptyset$ and $\mathcal{V}(C') \cap (\gamma_2 \cup \gamma_2') = \emptyset$.

  The first step is to apply the lemma 3.5i which allow us to assume that γ_1' and γ_2' only contains fresh variables (not free in A, B, γ_1, γ_2 and $\mathcal{V}(\sigma)$).

  Now, we can apply the induction hypothesis and we find a substitution σ and a constraints set γ_0 such that $A''\sigma \subset_{\gamma_0} C''$ with $\gamma_0 - \gamma_2$ containing only fresh variables. We can also apply it to find another substitution σ' and a constraint set γ_0' such that $C'\sigma' \subset_{\gamma_0'} A'$ with $\gamma_0' - \gamma_1'$ containing only fresh variables. Moreover we have for all $X \notin \gamma_1, X\sigma = X$ and for all $X \notin \gamma_2', X\sigma' = X$.

  From the latest fact and because variables in γ_2' are fresh, we find $C'\sigma' = C'$, which give $C' \subset_{\gamma_0'} A'$. Moreover, we also get $\mathcal{V}(C') \cap (\gamma_0 \cup \gamma_0') = \emptyset$ from $\mathcal{V}(C') \cap (\gamma_2 \cup \gamma_2') = \emptyset$ (this is true because γ_1', $\gamma_0 - \gamma_2$ and $\gamma_0' - \gamma_1'$ contain only fresh variables). Thus, we can apply again the lemma 3.4 to find $C'\sigma \subset_{\gamma_0'} A'\sigma$ (because by hypothesis, γ_0' is fresh enough). Then we can use the fact that $C'\sigma = C'$ because we applied first the lemma 3.5 to have $\gamma_1 \cap \mathcal{V}(C) = \emptyset$. So we now have

  - $C' \subset_{\gamma_0'} A'\sigma \quad$ and $\quad A''\sigma \subset_{\gamma_0} C''$.
  - $\mathcal{V}(C') \cap (\gamma_0 \cup \gamma_0') = \emptyset$.
  - for all $X \notin \gamma_1, X\sigma = X$.
  - $\gamma_0 - \gamma_2$ contains only fresh variables.

  And this give the wanted result by applying the $\rightarrow^{\subset}$ rule.

  In this case we add one rule to replace two rules. So using the induction hypothesis, the number of rules decreases.

- If the last rule for the hypothesis $A \subset_{\gamma_1} B$ is the $\forall_r^{\subset}$ rule and the last rule for the hypothesis $B \subset_{\gamma_2} C$ is the $\forall_l^{\subset}$ rule, then we have $B = \forall Y B'$, $\gamma_1 = \gamma_1' \cup \{Y\}$ and we have deduced the two hypothesis from $A \subset_{\gamma_1'} B'$, $B'[Y \Leftarrow D] \subset_{\gamma_2} C$. If $Y \notin \mathcal{V}(B)$ then the result is immediate by induction. Otherwise, We can apply lemma 3.4 to find $A[Y \Leftarrow D] \subset_{\gamma_1'} B'[Y \Leftarrow D]$ (because we can assume $Y \notin \gamma_1'$ using corollary 3.3 and because we get $\mathcal{V}(D) \cap \gamma_1' = \emptyset$ from the lemma 3.2). Then we can apply the induction hypothesis and we

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

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

$$\frac{\Gamma \vdash \mathtt{t} : A \qquad A \subset_\gamma B \qquad \mathcal{V}(\Gamma) \cap \gamma = \emptyset}{\Gamma \vdash \mathtt{t} : B}\, \subset$$

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

find σ such that $A[Y \Leftarrow D]\sigma \subset_{\gamma_0} C$ with for all $X \notin \gamma_1, X\sigma = X$ and $\gamma_0 - \gamma_2$ contains only fresh variables.

In this case we add no rule to replace two rules. So using the induction hypothesis, the number of rules decreases. ■

Properties of the type-system. the typing rules of the system $\mathbf{F}^\eta$ are given in table 3. One can remark that there are no quantification rule among the typing rules because the sub-typing alone can handle all the operations on the quantification as shown by the next lemma.

Lemma 3.7 *If* $\Gamma \vdash_F \mathtt{t} : A$ *then* $\Gamma \vdash \mathtt{t} : A$.

Proof: Using the $Ax^\subset$ and $\forall_r^\subset$ rules, we get that $\forall X\, A \subset_\emptyset A[X \Leftarrow B]$. Similarly, $A \subset_{\{X\}} \forall X\, A$. 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 $\Gamma \vdash_F \mathtt{t} : A$. ■

Lemma 3.8 *If* $\Gamma \vdash \mathtt{t} : F$ *then we can find* $\mathtt{t}' \in \Lambda$ *such that* $\mathtt{t}' \triangleright_\eta \mathtt{t}$ *and* $\Gamma \vdash_F \mathtt{t}' : F$.

Proof: The proof is by induction on the structure of the derivation of $\Gamma \vdash \mathtt{t} : F$. The only non-trivial case is the $\subset$ rule, but this case follows from lemma 3.1. ■

Theorem 3.9 *Strong normalisation holds for system* $\mathbf{F}^\eta$.

Proof: This result follows from lemma 3.8 and the strong normalisation of system *=[[**?**]]**F** [5]. ■

Lemma 3.10 *If* $\Gamma \vdash t : A$, *then for all substitution* σ, $\Gamma\sigma \vdash t : A\sigma$. *Moreover, the structure 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 $\Gamma \vdash t : B$, $B \subset_\gamma A$ and $\mathcal{V}(\Gamma) \cap \gamma = \emptyset$. First, we apply the lemma 3.5 to get a fresh γ' (such that σ fulfills the hypothesis of the lemme 3.4). This gives us $B\sigma' \subset_{\gamma'} A$. Then we can apply the induction hypothesis to get $\Gamma \vdash t : B\sigma'$ (because $\mathcal{V}(\Gamma) \cap \gamma = \emptyset$ implies $\Gamma\sigma' = \Gamma$). Finally, as γ' contains only fresh variables, we can apply lemma 3.4 to find $B\sigma'\sigma \subset_{\gamma'} A\sigma$ followed by the induction hypothesis to find $\Gamma\sigma \vdash t : B\sigma'\sigma$. This gives the wanted result if we use the $\subset$ rule. The structure of the proof is left unchanged because applying lemmas 3.4 and 3.5 did not change it. ■

Lemma 3.11 *If $\Gamma \vdash t : A$, then there is a proof with the same conclusion sequent which do not use two succesive 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:

$$\dfrac{\dfrac{\Gamma \vdash t : A \qquad A \subset_\gamma B \qquad \mathcal{V}(\Gamma) \cap \gamma = \emptyset}{\Gamma \vdash t : B}\ \subset \qquad B \subset_{\gamma'} C \qquad \mathcal{V}(\Gamma) \cap \gamma' = \emptyset}{\Gamma \vdash t : C}\ \subset$$

Then we can use lemma 3.6 to find γ'' and σ such that $A\sigma \subset_{\gamma''} C$, for all $X \notin \gamma$, $X\sigma = X$ and $(\gamma'' - \gamma') \cap \mathcal{V}(\Gamma) = \emptyset$. So we have $\Gamma\sigma = \Gamma$ and we can apply lemma 3.10 to find $\Gamma \vdash t : A\sigma$. Then we can apply the $\subset$ rule to get the following sub-proof:

$$\dfrac{\Gamma \vdash t : A\sigma \qquad A\sigma \subset_{\gamma''} C \qquad \mathcal{V}(\Gamma) \cap \gamma'' = \emptyset}{\Gamma \vdash t : C}\ \subset$$

Moreover, because applying lemma 3.6 and lemma 3.10 does not increase the size of the proof, we can apply this process indefinitely to any proof (this make the size of the proof decrease at least by one for each step). When this process ends the final proof does not use two succesive instances of the $\subset$ rule. ■

Lemma 3.12 *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 induciton on the derivation. Here again the only non trivial case is the case of the $\subset$ rule. In this case, we have $\Gamma \vdash t : B$, $B \subset_\gamma A$ and $\mathcal{V}(\Gamma) \cap \gamma = \emptyset$. Let us write $\gamma = \{X_1, \ldots, X_n\}$ and choose $\gamma' = \{Y_1, \ldots, Y_n\}$ such that for all $i \in \{1, \ldots, n\}$, $Y_i \notin \mathcal{V}(A) \cup \mathcal{V}(B) \cup \mathcal{V}(C) \cup \mathcal{V}(\Gamma)$. Thus we can apply lemma 3.5 with $\sigma = [X_1, \ldots, X_n \Leftarrow Y_1, \ldots, Y_n]$ to find $B\sigma \subset_{\gamma'} A$. Then we have $(\mathcal{V}(\Gamma) \cup \mathcal{V}(C)) \cap \gamma' = \emptyset$ and using the lemma 3.10 and the induction hypothesis, we find $\Gamma, x : C \vdash t : B\sigma$ which gives the wanted result using the $\subset$ rule.

Lemma 3.13 *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 3.12. ■

Lemma 3.14 *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. ■

Theorem 3.15 *Subject reduction for $\beta\eta$-reduction hold in system $\mathbf{F}^\eta$.*

Proof: We want to prove that if $\Gamma \vdash t : A$ ans $t \triangleright_{\beta\eta} t'$ then $\Gamma \vdash t' : A$. As in [8], the proof goes by induction on the length of the derivation and for each reduction 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 3.11, we have a proof of the following shape:

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

The proof of $A \to B \subset_\gamma C \to D$ necessarely ends by the $\to^\subset$ or $Ax^\subset$ rules. In both cases we have $C \subset_{\gamma'} A$, $B \subset_\gamma D$ and $\mathcal{V}(C) \cap (\gamma \cup \gamma') = \emptyset$. Moreover, using the lemma 3.5i we can assume $\mathcal{V}(\Gamma) \cap \gamma' = \emptyset$. Then by using the $\subset$ rule we find $\Gamma \vdash u : A$ from $\Gamma \vdash u : C$. Thus using lemma 3.13, 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 3.11, we have a proof of the following shape with $\Gamma' = \Gamma, x : A$:

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

We know that $\lambda x\, (t\ x) \triangleright_\eta t$. So x does not appear free in t. Using lemma 3.14, we get $\Gamma \vdash t : C \to D$. Then we can produce the following proof to get the wanted result (because $\mathcal{V}(\Gamma') \cap \gamma' = \emptyset$ and $\mathcal{V}(\Gamma') \cap \gamma = \emptyset$ imply $\mathcal{V}(A) \cap (\gamma \cup \gamma') = \emptyset$):

$$\dfrac{\dfrac{\vdots}{\Gamma \vdash t : C \to D} \qquad \dfrac{\dfrac{\vdots}{A \subset_{\gamma'} C} \qquad \dfrac{\vdots}{D \subset_\gamma E} \qquad \mathcal{V}(A) \cap (\gamma \cup \gamma') = \emptyset}{C \to D \subset_\gamma A \to E}\ \to^\subset}{\Gamma \vdash t : A \to E}\ \subset$$

■

Theorem 3.16 *$\Gamma \vdash \mathtt{t} : F$ iff exists $\mathtt{t}' \in \Lambda$ such that $\mathtt{t}' \triangleright_\eta \mathtt{t}$ and $\Gamma \vdash_F \mathtt{t}' : F$.*

Proof: This lemma follows from lemma 3.7 and 3.8 and from theorem 3.15. ■

4 A partial typechecking algorithm.

4.1 Preliminaries and notation.

We now describe a partial algorithm for system $\mathbf{F}^\eta$. In order to be able to add type information to a term, we define the following notions:

Definition 4.1 *The set of terms with type information Λ^T is the smallest set such that:*

- $x \in \Lambda^T$ *if* $x \in \mathcal{V}_\Lambda$.
- $(t\,u) \in \Lambda^T$ *if* $t, u \in \Lambda^T$.
- $\lambda x\, t \in \Lambda^T$ *if* $x \in \mathcal{V}_\Lambda$ *and* $t \in \Lambda^T$ *(x is bound in $\lambda x\, t$).*
- $\lambda x : A\ \ t \in \Lambda^T$ *if* $x \in \mathcal{V}_\Lambda$, $t \in \Lambda^T$ *and* $A \in \mathcal{F}$ *(x is bound in $\lambda x : A\ \ t$).*
- $t : A \in \Lambda^T$ *if* $t \in \Lambda^T$ *and* $A \in \mathcal{F}$.
- $\Lambda X\, t \in \Lambda^T$ *if* $t \in \Lambda^T$ *and* $X \in \mathcal{V}_\mathcal{F}$ *(X is bound in $\Lambda X\, t$).*

Definition 4.2 *If $t \in \Lambda^T$ we define $t^* \in \Lambda$ as the λ-term obtained by erasing all type information (the formal definition is left to the reader).*

The reader should notice the main differences with the type annotation in Church's style: there is no type application. However, there is two possibilities to give type indication ($\lambda x : A\ \ t$ to indicate the type of the variable x or $t : A$ to indicate the type of the term t). Finally the annotation $\Lambda X\, t$ does not have the same meaning: in Church's style it denotes the introduction of a quantifier. Here it introduces a fresh type variable that could be introduced here but might be syntactically introduced at another place. It is also important to note that for the completeness theorem 4.20, one only need to use type annotation of the form $t : A$. The two other forms are only introduced because they are useful in practice.

To define the algorithm, we need to distinguish two kinds of variables: the unification variables and the constants. Thus we choose a partition of the set $\mathcal{V}_\mathcal{F} = \mathcal{V}^?_\mathcal{F} \oplus \mathcal{V}^!_\mathcal{F}$ in two disjoint infinite sets. Elements of the set $\mathcal{V}^?_\mathcal{F}$, are called unification variables and are written preceded by a question mark ($?X$). Elements of the set $\mathcal{V}^!_\mathcal{F}$, are called constants and are written with no mark (X). From now, all substitutions will only affect unification variables. Moreover, we will often write variables for unification variables. However, $\mathcal{V}(A)$ is still the set of all variables free in A (constants and unification variables together).

The algorithm manipulates elements of the following sets:

- $\mathrm{Ctx} = \mathcal{V}_\lambda \to_f \mathcal{F}$: the set of all contexts.
- $\Sigma = \mathcal{V}^?_\mathcal{F} \to_f \mathcal{F}$: the set of finite substitutions.
- $\mathcal{C} = \mathcal{P}^f(\mathcal{V}^!_\mathcal{F})$: the set of all set of constraints,
- $\mathcal{M} = \mathcal{V}^?_\mathcal{F} \to_f \mathcal{P}^f(\mathcal{V}^!_\mathcal{F})$: the set of *constraints mappings.*

A constraints mapping Θ expresses that constants in $\Theta(?X)$ should not appear free in the *value* of $?X$. If Θ and Θ' are sets of constraints, we write $\Theta \subset \Theta'$ if for all $?X \in \mathcal{V}^?_\mathcal{F}$, $\Theta(?X) \subset \Theta'(?X)$. Here are few definitions and lemmas concerning substitutions:

Definition 4.3 *We will say that a substitution $\sigma \in \Sigma$ is compatible with a constraints mapping $\Theta \in \mathcal{M}$ if for all unification variables $?X \in \mathcal{V}^?_\mathcal{F}$ we have $\mathcal{V}(?X\sigma) \cap \Theta(?X) = \emptyset$ (this reflect the meaning we intended for the constraints mapping Θ). We write this $\sigma \ll \Theta$.*

Definition 4.4 *We will say that a substitution $\sigma \in \Sigma$ is hereditary compatible with a constraints mapping $\Theta \in \mathcal{M}$ if $\sigma \ll \Theta$ and for all $?X, ?Y \in \mathcal{V}^?_\mathcal{F}$, $?Y \in \mathcal{V}(?X\sigma)$ implies $\Theta(?X) \subset \Theta(?Y)$. We write this $\sigma \lll \Theta$.*

Lemma 4.5 *If $\sigma \ll \Theta$ and $\Theta' \subset \Theta$ then $\sigma \ll \Theta'$.*

Lemma 4.6 *If $\sigma \ll \Theta$ (resp. $\lll$) and $\sigma' \lll \Theta$ then $\sigma \circ \sigma' \ll \Theta$ (resp. $\lll$).*

Proof: For both lemmas, the proof is a consequence of the definitions. ∎

4.2 The sub-typing algorithm.

The first function $\Pi \in \mathcal{P}^f(\mathcal{F}) \to \mathcal{C} \to \mathcal{M} \to \mathcal{M}$ used by the algorithm is the *propagation* of a set of constraints. $\Pi(A, \gamma, \Theta)$ extend Θ to make sure that the constants in γ are not used in the formula A: it tests that no constants in γ appears in A and adds γ to set of constants the variable $?X$ should not use, if and only if $?X$ appears free in A. It is defined as follows:

$\Pi(A, \gamma, \Theta) = \Theta'$

where $\mathcal{V}(A) \cap \gamma = \emptyset$ (otherwise Π is undefined)
for all $?X \in \mathcal{V}(A)$, $\Theta'(?X) = \Theta(?X) \cup \gamma$
for all $?X \notin \mathcal{V}(A)$, $\Theta'(?X) = \Theta(?X)$

Lemma 4.7 *If* $\Theta' = \Pi(A, \gamma, \Theta)$ *then* $\Theta \subset \Theta'$

Lemma 4.8 *If* $\Theta' = \Pi(A, \gamma, \Theta)$ *then for all* $\sigma \ll \Theta'$ *we have* $\gamma \cap \mathcal{V}(A\sigma) = \emptyset$.

Proof: The two previous lemmas are consequences of the definitions of Π and compatible. ■

Lemma 4.9 *If* $\Theta_1 = \Pi(A, \gamma, \Theta_0)$ *then for all substitution* σ *such that* $\sigma \ll \Theta_0$ *(resp.* $\lll$*) and* $A\sigma = A$*, we have* $\sigma \ll \Theta_1$ *(resp.* $\lll$*).*

Proof: This is a consequence of the definition of Π and compatible (we need to distinguish two cases depending if the variable we consider occurs free in A or not). ■

Lemma 4.10 *If* $\Theta_1 = \Pi(A, \Theta(?X), \Theta_0)$ *and* $?X \notin \mathcal{V}(A)$*, then for all substitution* σ *such that* $\sigma \lll \Theta_0$*,* $?X\sigma = ?X$ *and* $A\sigma = A$*, we have* $[?X \Leftarrow A] \circ \sigma \lll \Theta_1$.

Proof: Using the lemma 4.6 we just need to prove $[?X \Leftarrow A] \lll \Theta_1$ and $\sigma \lll \Theta_1$. The second is a direct consequence of lemma 4.9 and the first is a consequence of the definitions.

We can now define the sub-typing algorithm $S(A, B, \Theta)$ where $A, B \in \mathcal{F}$ are types and $\Theta \in \mathcal{M}$ is a constraints mapping. The result of S is a triple $(\sigma, \gamma, \Theta')$, where $\sigma \in \Sigma$, $\gamma \in \mathcal{C}$ and $\Theta' \in \mathcal{M}$. S is the partial function defined by applying the first possible rule among the following:

$S(A, A, \Theta) = \epsilon, \emptyset, \Theta$ (0)

$S(A \to B, C \to D, \Theta) = \sigma_1 \circ \sigma_0, \gamma_1, \Theta_2$ (1)
where $\sigma_0, \gamma_0, \Theta_0 = S(C, A, \Theta)$
$\sigma_1, \gamma_1, \Theta_1 = S(B\sigma_0, D\sigma_0, \Theta_0)$
$\Theta_2 = \Pi(C\sigma_0\sigma_1, \gamma_1 \cup \gamma_0, \Theta_1)$.

$S(A, ?X, \Theta) = \sigma_1, \emptyset, \Theta_1$ (2)
where $?X \notin \mathcal{V}(A)$
$\sigma_1 = [?X \Leftarrow A]$
$\Theta_1 = \Pi(A, \Theta(?X), \Theta)$

$S(?X, A, \Theta) = \sigma_1, \emptyset, \Theta_1$ (3)
where $?X \notin \mathcal{V}(A)$
$\sigma_1 = [?X \Leftarrow A]$
$\Theta_1 = \Pi(A, \Theta(?X), \Theta)$

$S(\forall X\, A, B, \Theta) = S(A[X \Leftarrow ?Y], B, \Theta)$ (4)
where $?Y$ is a fresh variable.

$S(A, \forall X\, B, \Theta) = \sigma_1, \gamma_1, \Theta_2 \cup \{Y\}$ (5)
where Y is a fresh constant
$\Theta_1 = \Pi(\forall X\, B, \{Y\}, \Theta)$
$\sigma_1, \gamma_1, \Theta_2 = S(A, B[X \Leftarrow Y], \Theta_1)$

This definition can be partial in three ways: there might be no possible rule to apply, the function Π might be undefined or the definition might loop. The typical case of a loop is with $S(F, G, \Theta)$ where $F = \forall X(X \to (X \to X \to Y) \to Y)$ and $G = F \to F \to Y$.

In the rule (4) and (5), we need to choose a fresh constant or unification variable. Fresh means that it does not occur in any formula which was build before the fresh variables or constants were created. A way to formalise this is to add one more parameter to S to remember all the previously used variables and constants.

Lemma 4.11 *If $S(A, B, \Theta) = \sigma, \gamma, \Theta'$ then $\Theta \subset \Theta'$*

Proof: An induction on the computation of $S(A, B, \Theta)$ gives immediately the result using lemma 4.7, because Θ' is modified only by applying Π.

Lemma 4.12 *If $S(A, B, \Theta) = \sigma_1, \gamma, \Theta'$ then for all substitution $\sigma_0 \lll \Theta$ and such that $A\sigma_0 = A$ and $B\sigma_0 = B$, we have $\sigma_1 \circ \sigma_0 \lll \Theta'$.*

Proof: We prove this by induction on the computation of $S(A, B, \Theta) = \sigma_1, \gamma, \Theta'$. The proof uses lemma 4.9 for the rule (1) and (5) and lemma 4.10 for the rule (2) and (3). ■

Lemma 4.13 *If $S(A, B, \Theta) = \sigma, \gamma, \Theta'$ then for all $\sigma' \ll \Theta'$, $A\sigma\sigma' \subset_\gamma B\sigma\sigma'$.*

Proof: We prove this by induction using an induction on the sequence of rules applied to compute S. We use the notation of the definition of S and we distinguish the following cases:

- If the last applied rule is (1). Using lemmas 4.5, 4.6, 4.11 and 4.12 we find $\sigma_0 \lll \Theta_0$, $\sigma_1 \lll \Theta_1$, $\sigma_1 \circ \sigma_0 \lll \Theta_1$, $\sigma' \ll \Theta_1$ and $\sigma' \circ \sigma_1 \ll \Theta_0$. Thus we can apply the induction hypothesis to find $B\sigma_0\sigma_1\sigma' \subset_{\gamma_0} D\sigma_0\sigma_1\sigma'$ and $C\sigma_0\sigma_1\sigma' \subset_{\gamma_1} A\sigma_0\sigma_1\sigma'$. Then, using lemma 4.8, from $\sigma' \ll \Theta_2 = \Pi(\{C\sigma_0\sigma_1\}, \gamma_0 \cup \gamma_1, \Theta_1)$, we deduce $\mathcal{V}(C\sigma_0\sigma_1\sigma') \cap (\gamma_0 \cup \gamma_1) = \emptyset$. So we can apply the $\rightarrow^{\subset}$ rule to find the expected result: $(A \rightarrow B)(\sigma_1 \circ \sigma_0)\sigma' \subset_{\gamma_0} (C \rightarrow D)(\sigma_1 \circ \sigma_0)\sigma'$
- If the last applied rule is (2),(3) or (0), then the result is immediate using the $Ax^{\subset}$ rule.
- If the last applied rule is (4), we have $S(\forall X\, A, B, \Theta) = S(A[X \Leftarrow ?Y], B, \Theta) = \sigma_1, \gamma_1, \Theta_1$ and $\sigma' \ll \Theta_1$. The induction hypothesis gives $A[X \Leftarrow ?Y]\sigma_1\sigma' \subset_{\gamma_1} B\sigma_1\sigma'$. X can be chosen to have $X\sigma_1\sigma' = X$ and $X \notin \mathcal{V}(\sigma' \circ \sigma_1)$. Thus we get $A\sigma_1\sigma'[X \Leftarrow ?Y\sigma_1\sigma'] \subset_{\gamma_1} B\sigma_1\sigma'$ and we get the expected result, using the $\forall_l^{\subset}$ rule, because $\forall X\, (A\sigma_1\sigma') = (\forall X\, A,)\sigma_1\sigma'$.
- If the last applied rule is (5), we have $S(A, \forall X\, B, \Theta) = (\sigma_1, \gamma_1, \Theta_2 \cup \{Y\})$ where Y is a fresh constant, $(\Theta_1, \sigma_1, \gamma_1) = S(A, B[X \Leftarrow Y], \Theta)$ and $\sigma' \ll \Theta_1$. Then the induction hypothesis gives $A\sigma_1\sigma' \subset_{\gamma_1} B[X \Leftarrow Y]\sigma_1\sigma'$. Then, using the $\forall_l^{\subset}$ rule, we find $A\sigma_1\sigma' \subset_{\gamma_1 \cup \{Y\}} \forall Y\, (B[X \Leftarrow Y]\sigma_1\sigma')$. This gives the wanted result because Y does not occur free in $(\forall X\, B)\sigma_1\sigma'$ (because of the call to Π) and because X can be chosen arbitrarely to have $\forall Y\, (B[X \Leftarrow Y]\sigma_1\sigma') = (\forall X\, B)\sigma_1\sigma'$. ■

Lemma 4.14 *If there exists a most general unifier (m.g.u.) σ for $A, B \in \mathcal{F}$ and if $\sigma \lll \Theta$, then $S(A, B, \Theta) = \sigma', \gamma, \Theta'$ where γ is a set of fresh constants and where for all unification variables $?X$ such that $?X\sigma' \neq ?X$, either $?X\sigma' = ?X\sigma$, either $?X$ is a fresh unification variable (not free in A and B or $\mathcal{V}(\sigma)$).*

Proof: The proof is by induction on the construction of the formulas A and B. If they are equal formulas or if one of them is a variable, then the rule (0),(2) or (3) give directly the result. Otherwise they must both start by an implication or a quantifier. In both cases the induction hypothesis gives immediately the result (in the case of quantifiers, we use rule (4) and (5) and in the case of implications, we use the property that if σ is the m.g.u. of $A \rightarrow B$ and $C \rightarrow D$ then $\sigma = \sigma_1 \circ \sigma_0$, where σ_0 is the m.g.u. of B and D and σ_1 is the m.g.u. of $A\sigma_0$ and $C\sigma_0$). ■

Lemma 4.15 *If $A \subset_\gamma B$, with no unification variables free in A or B, then there exists a number p and p formulas $C_1, \ldots, C_p$ with no free unification variables, such that for all $\Theta \in \mathcal{M}$, $S(A, C_1, \Theta) = \sigma_0, \gamma_0, \Theta_0$, $S(C_1, C_2, \Theta) = \sigma_1, \gamma_1, \Theta_1$, $\ldots$, $S(C_p, B, \Theta) = \sigma_p, \gamma_p, \Theta_p$. Moreover, for all $Y \in \gamma_0 \cup \ldots \cup \gamma_p$, Y is fresh or $Y \in \gamma$.*

Proof: First, we can assume $\mathcal{V}(A) \cap \gamma = \emptyset$. Indeed, $A \subset_\gamma B$ implies $A \subset_{\{X_1,\ldots,X_n\}} \forall X_1 \ldots \forall X_n\, A$ and $\forall X_1 \ldots \forall X_n\, A \subset_\gamma B$ with $\{X_1, \ldots, X_n\} = \mathcal{V}(A) \cap \gamma$. Therefore the result follows from an induction on the proof of $A \subset_\gamma B$, using $\mathcal{V}(A) \cap \gamma = \emptyset$ as an extra hypothesis.

- The result is obvious in the case of the $Ax^\subset$ rule (we can choose $p = 0$ and $\gamma_0 = \gamma = \emptyset$) .
- If the last rule is the $\rightarrow^\subset$ rule. We have $A = A' \rightarrow A''$, $B = B' \rightarrow B''$, $A'' \subset_\gamma B''$, $B' \subset_\delta A'$ and $\mathcal{V}(B') \cap (\gamma \cup \delta) = \emptyset$. By ind. hyp. we can find two naturals p and q, p formulas $C_1, \ldots, C_p$ such that $S(B', C_1, \Theta) = \sigma_0, \gamma_0, \Theta_0$, $S(C_1, C_2, \Theta) = \sigma_1, \gamma_1, \Theta_1$, $\ldots$, $S(C_p, A', \Theta) = \sigma_p, \gamma_p, \Theta_p$ and q formulas $D_1, \ldots, D_q$ such that $S(A'', D_1, \Theta) = \sigma'_0, \gamma'_0, \Theta'_0$, $S(D_1, D_2, \Theta) = \sigma'_1, \gamma'_1, \Theta'_1$, $\ldots$, $S(D_q, B'', \Theta) = \sigma'_q, \gamma'_q, \Theta'_q$. Then we can consider $E_1 = A' \rightarrow D_1$, $\ldots$, $E_q = A' \rightarrow D_q$, $E_{q+1} = C_p \rightarrow B''$, $E_{q+2} = C_{p-1} \rightarrow B''$, $\ldots$, $E_{q+2} = C_1 \rightarrow B''$. Thus we get the wanted results by applying the rule (1) because any constants in $\gamma_1, \ldots, \gamma_p, \gamma'_1, \ldots, \gamma'_q$ is either fresh or occur in γ or δ.
- If the last rule is $\forall_l^\subset$, we have $A = \forall X\, A'$ and $A'[X \Leftarrow C] \subset_\gamma B$. Then we get the wanted result by adding $A'[X \Leftarrow C]$ at the beginning of the formulas produced when we apply the induction hypothesis to $A'[X \Leftarrow C] \subset_\gamma B$. This works, because the lemma 4.14 tells us that for any fresh variable $?X$, $S(A'[X \Leftarrow ?X], A'[X \Leftarrow C], \Theta) = [?X \Leftarrow C] \circ \sigma', \gamma', \Theta'$, with all the constant in γ' fresh ($[?X \Leftarrow C]$ is the m.g.u. of $A'[X \Leftarrow C]$, $A'[X \Leftarrow ?X]$ and is compatible with Θ because $?X$ is fresh which implies $\Theta(X) = \emptyset$). Thus using the rule (2) we deduce $S(A, A'[X \Leftarrow C], \Theta) = [?X \Leftarrow C] \circ \sigma', \gamma', \Theta'$.
- If the last rule is $\forall_r^\subset$, we have $B = \forall X\, B'$, $\gamma = \gamma' \cup \{X\}$ and we deduced $A \subset_\gamma B$ from $A \subset_{\gamma'} B'$. Then the induction hypothesis gives the expecting result, because we assumed $\mathcal{V}(A) \cap \gamma = \emptyset$. Thus for any fresh constant Y, we have $A \subset_{\gamma'} B'[X \Leftarrow Y]$, which gives the wanted result because for all formulas C_p, $S(C_p, B, \Theta) = \sigma_p, \gamma_p \cup \{Y\}, \Theta_p$ where Y is a fresh constant and where $\sigma_p, \gamma_p, \Theta_p = S(C_p, B'[X \Leftarrow Y], \Theta)$.

■

Lemma 4.16 *If $A, B \in \mathcal{F}$ are quantifier free and if $\Theta \in \mathcal{M}$ verifies $\Theta(?X) = \emptyset$ for all $?X \in \mathcal{V}_\mathcal{F}$, then A and B unify if and only if $S(A, B, \gamma, \Theta)$ is defined (if they do not unify, the computation does not loop). Moreover, if $S(A, B, \Theta) = (\Theta, \sigma, \emptyset)$, then σ is the m.g.u. of A and B.*

Proof: If A and B are quantifier free then the rules (4) and (5) are never used. Thus the algorithm is exactly an unification algorithm (γ stays empty, the value of Θ is never changed and the rules (2) and (3) perform the occur-check we need in a standard unification algorithm).

4.3 The typing algorithm.

The typing algorithm is in fact a type checking algorithm which can also be used for type inference, because we can always call it with a unification variable. It is a partial function $T \in \mathrm{Ctx} \rightarrow \Lambda^T \rightarrow \mathcal{F} \rightarrow \mathcal{C} \rightarrow (\Sigma \times \mathcal{C})$ defined as follows ($\mathcal{F}(\Gamma)$ denotes a formula with the same free variables as Γ, for instance if $\Gamma = x_1 : A_1, \ldots, x_n : A_n$, one can choose $\mathcal{F}(\Gamma) = A_1 \rightarrow \ldots \rightarrow A_n$):

$T(\Gamma, x, A, \Theta) = \sigma, \Theta_2$ (0)
where $x : B \in \Gamma$
$\sigma, \gamma, \Theta_1 = S(B, A, \Theta)$
$\Theta_2 = \Pi(\mathcal{F}(\Gamma\sigma), \gamma, \Theta_1)$

$T(\Gamma, (tu), A, \Theta) = (\sigma_2 \circ \sigma_1), \Theta_2$ (1)
where $?X$ is a fresh variable
$\sigma_1, \Theta_1 = T(\Gamma, t, (?X \to A), \Theta)$
$\sigma_2, \Theta_2 = T(\Gamma\sigma_1, u\sigma_1, ?X\sigma_1, \Theta_1)$

$T(\Gamma, \lambda x\, t, A, \Theta) = (\sigma_3 \circ \sigma_1), \Theta_3$ (2)
where $?X, ?Y$ are fresh variables
$\sigma_1, \gamma, \Theta_1 = S(?X \to ?Y, A, \Theta_1)$
$\Theta_2 = \Pi(\mathcal{F}(\Gamma\sigma_1), \gamma, \Theta_1)$
$\sigma_3, \Theta_3 = T((\Gamma, x : ?X)\sigma_1, t\sigma_1, ?Y\sigma_1, \Theta_2)$

$T(\Gamma, \lambda x : B\, t, A, \Theta) = (\sigma_3 \circ \sigma_1), \Theta_3$ (3)
where $?Y$ is a fresh variable
$\sigma_1, \gamma, \Theta_1 = S(B \to ?Y, A, \Theta_1)$
$\Theta_2 = \Pi(\mathcal{F}(\Gamma\sigma_1), \gamma, \Theta_1)$
$\sigma_3, \Theta_3 = T((\Gamma, x : B)\sigma_1, t\sigma_1, ?Y\sigma_1, \Theta_2)$

$T(\Gamma, t : B, A, \Theta) = (\sigma_3 \circ \sigma_1), \Theta_3$ (4)
where $\sigma_1, \gamma, \Theta_1 = S(B, A, \Theta_1)$
$\Theta_2 = \Pi(\mathcal{F}(\Gamma\sigma_1), \gamma, \Theta_1)$
$\sigma_3, \Theta_3 = T(\Gamma\sigma_1, t\sigma_1, B\sigma_1, \Theta_2)$

$T(\Gamma, \Lambda X\, t, A, \Theta) = \sigma_2, \Theta_2$ (5)
where Y is a fresh constant
$\Theta_1 = \Pi(\mathcal{F}(\Gamma), \{Y\}, \Theta)$
$\sigma_2, \Theta_2 = T(\Gamma, t[X \Leftarrow Y], A, \Theta_1)$

Note: The treatment of $\Lambda X\, t$ in rule (5) could seem surprising. In fact this construction is only used to introduce a name X local to a term t. It is not directly related to an introduction of a quantifier. Moreover the call to Π in this rule is not necessary for the correctness, but it makes the rule more natural when we add the generalisation rule (see section 4.4).

One should also remark that the term is read form left to right (in the rule (1) the function is typechecked before its argument. For the following results this is not relevant. However, when an error occurs while typing a sub-term of a term t, one know that it is only useful to add type information inside this sub-term or to its left. This is very important when writing program.

Lemma 4.17 *If* $T(\Gamma, t, A, \Theta) = \sigma, \Theta'$ *then* $\Theta \subset \Theta'$

Proof: The proof is by induction on the computation of T and uses lemmas 4.7 and 4.11. ■

Lemma 4.18 *If* $T(\Gamma, t, A, \Theta) = \sigma, \Theta'$ *then* $\sigma \lll \Theta'$.

Proof: The proof is by induction on the computation of T and uses the lemmas 4.6 and 4.12.■

Theorem 4.19 (Correctness) *If* $T(\Gamma, t, A, \Theta) = \sigma, \Theta'$ *then for all* $\sigma' \ll \Theta'$, *we have* $\Gamma\sigma\sigma' \vdash t^* : A\sigma\sigma'$

Proof: We prove this result is by induction on the computation of T. We use the notation of the definition of T:

- If the last applied rule is (0), then we have $\sigma' \ll \Theta_1$ (using lemmas 4.5 and 4.7. Then, lemma 4.13 gives $B\sigma\sigma' \subset_\gamma A\sigma\sigma'$. Finally, we get $\gamma \cap \mathcal{V}(\Gamma\sigma\sigma') = \emptyset$ from lemma 4.8 and we get the expected result using the Ax and $\subset$ rule.
- If the last applied rule is (1), then we have $\sigma' \circ \sigma_2 \ll \Theta_1$ (using lemmas 4.5, 4.6 and 4.7). Then the induction hypothesis gives $\Gamma\sigma_1\sigma_2\sigma' \vdash u^* : ?X\sigma_1\sigma_2\sigma'$ and $\Gamma\sigma_1\sigma_2\sigma' \vdash t^* : (?X \to A)\sigma_1\sigma_2\sigma'$. Therefore we get the wanted result by applying the $\to_e$ rule.
- If the last rule is (2), then we have $\sigma' \circ \sigma_3 \ll \Theta_1$ and $\sigma' \circ \sigma_3 \ll \Theta_2$ (using lemmas 4.5, 4.6 and 4.7). Then the induction hypothesis gives $\Gamma\sigma_1\sigma_3\sigma', x : ?X\sigma_1\sigma_3\sigma' \vdash t^* : ?Y\sigma_1\sigma_3\sigma'$ and lemma 4.13 gives $(?X \to ?Y)\sigma_1\sigma_3\sigma' \subset_\gamma A\sigma_1\sigma_3\sigma'$. Moreover, lemma 4.8 implies that $\Gamma\sigma_1\sigma_3\sigma' \cap \gamma = \emptyset$. Therefore, we can apply the $\to_i$ and $\subset$ rules to find the expected result.
- The case of rule (3) is similar to the previous case.

- If the last rule is (4), then we have $\sigma' \circ \sigma_3 \ll \Theta_1$ and $\sigma' \circ \sigma_3 \ll \Theta_2$ (using lemmas 4.5, 4.6 and 4.7). Then the induction hypothesis gives $\Gamma\sigma_1\sigma_3\sigma' \vdash t^* : B\sigma_1\sigma_3\sigma'$ and lemma 4.13 gives $B\sigma_1\sigma_3\sigma' \subset_\gamma A\sigma_1\sigma_3\sigma'$. Moreover, lemma 4.8 implies that $\Gamma\sigma_1\sigma_3\sigma' \cap \gamma = \emptyset$. Therefore, we can apply the $\subset$ rule to find the expected result.

- If the last rule is (5), the induction hypothesis immediately gives the result.

■

Theorem 4.20 (Completeness) *If $\Gamma \in Ctx$, $A \in \mathcal{F}$ contains no unification variable and if for $t_0 \in \Lambda$ we have $\Gamma \vdash t_0 : A$, then there exists a term with type information $t \in \Lambda^T$ such that $t_0 = t^*$, and such that for all Θ we have $T(\Gamma, t, A, \Theta) = \sigma, \Theta'$.*

Proof: The proof is by induction on the derivation of $\Gamma \vdash t_0 : A$.

- If the last rule is the Ax rule, then we can take $t = t_0$.

- If the last rule is the $\rightarrow_i$ rule, then, using the induction hypothesis, from $\Gamma, x : A \vdash t_0 : B$, we can find a term $t \in \Lambda^T$ such that $t^* = t_0$ and $T((\Gamma, x : A), t, B, \Theta) = \sigma, \Theta'$. Then if we choose two fresh variables $?X$ and $?Y$, by lemma 4.14, we have $S(?X \rightarrow ?Y, A \rightarrow B, \Theta) = \sigma_0, \emptyset, \Theta$ with $\sigma_0 = [?X \Leftarrow A] \circ [?Y \Leftarrow B]$. So using the rule (3) we find $T(\Gamma, \lambda x\, t, A \rightarrow B, \Theta) = \sigma_0 \circ \sigma, \Theta'$.

- If the last rule is the $\rightarrow_e$ rule, then, using the induction hypothesis, from $\Gamma \vdash t_0 : A \rightarrow B$, and $\Gamma \vdash u_0 : A$ we find $t, u \in \Lambda^T$ such that $t^* = t_0$, $u^* = u_0$ and such that for all Θ, $T(\Gamma, u, A, \Theta) = \sigma_0, \Theta_0$ and $T(\Gamma, t, A \rightarrow B, \Theta_0) = \sigma_1, \Theta_1$. Thus if we choose a fresh variable $?X$, we have $S(A, ?X, \Theta) = [?X \Leftarrow A], \Theta$ (because $?X \notin \mathcal{V}(A)$). Thus we can apply the rule (4) to find $T(\Gamma, (u : A), ?X, \Theta) = [X \Leftarrow A] \circ \sigma_0, \Theta_0$. Finally, rule (1) gives $T(\Gamma, (t\ (u : A)), B, \Theta) = \sigma_1 \circ [X \Leftarrow A] \circ \sigma_0, \Theta_1$ (we use the fact that Γ, A and B are not affected by the substitution because they contain no unification variables).

- If the last rule is the $\subset$ rule, then, using the induction hypothesis, from $\Gamma \vdash t_0 : A$, we can find $t \in \Lambda^T$ such that $t^* = t_0$ and $T(\Gamma, t, A, \Theta) = \sigma, \Theta'$. Moreover, using lemma 4.15, from $A \subset_\gamma B$, we find some formulas $C_1, \ldots, C_p$ with no free unification variables such that for all $\Theta \in \mathcal{M}$, $S(C_p, B, \Theta) = \sigma_0, \gamma_0, \Theta_0$, $S(C_{p-1}, C_p, \Theta_0) = \sigma_1, \gamma_1, \Theta_1$, $\ldots$, $S(A, C_1, \Theta_{p-1}) = \sigma_p, \gamma_p, \Theta_p$. Moreover, we know that any constants in one of the γ_i is "fresh" or occur in γ, so they do not occur in Γ, which implies $\Pi(\mathcal{F}(\Gamma), \gamma_i, \Theta_i) = \Theta_i$. Thus we have $T(\Gamma, t : A : C_1 : \ldots : C_p, B, \Theta) = \sigma_p, \Theta_p$. ■

The rule (3) and (5) are not strictly necessary for this theorem (it only uses type annotations of the form $t : A$). However, in practice, type annotation of the form $\lambda x : A\, t$ or $\Lambda X\, t$ are very useful.

Theorem 4.21 *If t is a term with no type information and if $\Theta(?Y) = \emptyset$ for all $?Y \in \mathcal{V}^?_{\mathcal{F}}$, then t is typeable in simply-typed lambda-calculus if and only if $T(\emptyset, t, ?X, \Theta) = \sigma, \Theta$ is defined. Moreover, if defined, $?X\sigma$ is the most general type for t.*

Proof: When t is free of type information and when the context is free of quantifiers, only the rule (0), (1) and (2) are used, Θ is never modified and by lemma 4.16 S acts like a unification algorithm. Therefore, the computation of T is exactly Milner's algorithm, which implies the proposition.

4.4 The generalisation rule.

The proof of the theorem 4.21 tells us that our algorithm is really an extension of Milner's type-inference algorithm. But this is not quite true because for simplicity we omitted the generalisation. However, it is easy to define a generalisation function G which given a context Γ, a constraints mapping Θ and a formula A introduces quantifiers for all the variables and constants which can not appear in the context. $G(\Gamma, A, \Theta)$ it can be define as follows:

$$\begin{aligned} G(\Gamma, A, \Theta) &= \forall ?X_1 \ldots \forall ?X_n \forall Y_1 \ldots \forall Y_p\, A \\ &\quad \text{where } \{?X_1, \ldots, ?X_n\} = (\mathcal{V}(A) \cap \mathcal{V}^?_{\mathcal{F}}) - \mathcal{V}(\Gamma) \\ &\quad \text{and } \{Y_1, \ldots, Y_n\} = \{X \in \mathcal{V}(A) \cap \mathcal{V}^!_{\mathcal{F}}; \Pi(\mathcal{F}(\Gamma), \{X\}, \Theta) = \Theta\} \end{aligned}$$

Note: The use of Π tells us that X will never appear free in the context. The order in which quantifications are introduced is irrelevant because the rule or sub-typing will behave the same whatever order we use. The call to Π in the rule (5) to compute T guaranty that the fresh variable Y which is introduced can be generalised (this make the meaning of $\Lambda X\, t$ quite similar to the usual one, the only difference is that the generalisation is not perform immediately).

This generalisation can be used associated to a *let* construction as in Milner's algorithm. This make our algorithm a real extension of his. However, we can also perform generalisation for any argument in an application node. In both cases, the proof of correctness and completeness are unchanged. In practise, the second approach seems to make little differences and it is more elegant because it treats the *let* constructor as any redex (but we loose "backward compatibility" with Milner's algorithm, which is important if we want to extend an existing language).

4.5 Experiments.

We have implemented the type-checking algorithm using the Caml-Light language. We have tested it on various examples: Church's numeral, polymorphic lists, existential types, streams, two error monads (to raise and catch errors), binary numbers, ... These terms requires more then rank 2 polymorphism [7] because they use purely functional encoding of data-types. However, very little type annotation is needed to type these terms and the time to type-check all of them is much less than a second on a Sparc station 10 (for 100 lines of code).

Here is the example of streams with enough type information to type check it (each definition is followed by the type inferred by the algorithm). A stream is represented by a tuple containing an internal *state*, a function to compute the next element of the stream from the state and another function to compute the next state. The type of streams uses a negative quantifier to make the state "private" to the stream.

The type of streams:
$\texttt{type}\ \mathbf{Stream}[A] = \forall X\, (\forall S\, (S \to\ (S \to S) \to\ (S \to A) \to X) \to X)\,;$
The constructor for streams:
$\texttt{def}\ \mathbf{pack} = \Lambda A \lambda s \lambda f \lambda a\ (\lambda g\ (g\ s\ f\ a))\ : \mathbf{Stream}[A];$
$\mathbf{pack} : \forall X \forall A\, (X \to\ (X \to X) \to\ (X \to A) \to \mathbf{Stream}[A])$
The two destructors:
$\texttt{def}\ \mathbf{head} = \Lambda A \lambda s : \mathbf{Stream}[A]\ (s\ (\lambda s_1 \lambda f \lambda a\ (a\ s_1)))\,;$
$\mathbf{head} : \forall X\, (\mathbf{Stream}[X] \to X)$
$\texttt{def}\ \mathbf{tail} = \Lambda A \lambda s : \mathbf{Stream}[A]\ (s\ (\lambda s_1 \lambda f \lambda a\ (\mathbf{pack}\ \ (f\ s_1)\ f\ a)))\,;$
$\mathbf{tail} : \forall X\, (\mathbf{Stream}[X] \to \mathbf{Stream}[X])$
Two examples using streams:
$\texttt{def}\ \mathbf{map} = \lambda s \lambda f\ (\mathbf{pack}\ s\ \mathbf{tail}\ \ (\lambda s\ (f\ (\mathbf{head}\ s))))\,;$
$\mathbf{map} : \forall X \forall Y\, (\mathbf{Stream}[X] \to\ (X \to Y) \to \mathbf{Stream}[Y])$
$\texttt{def}\ \mathbf{nats} =\ (\mathbf{pack}\ 0\ (\lambda x\ (\mathbf{add}\ x\ 1))\ \ (\lambda x\, x))\,;$
$\mathbf{nats} : \mathbf{Stream}[\mathbf{Nat}]$

To see more examples, or to try yourself the algorithm, you can download a "normaliser for pure and typed lambda-calculus", implementing it, from the following URL:
`http://www.logique.jussieu.fr/www.raffalli/normaliser.html`.

5 Conclusion

The results and experiments presented in this paper give real hope that our algorithm could work in practice. The next step is its integration into a full scale programming language. Indeed, we must test it on a larger scale to discover what are the problems we encounter to incorporate the other features of a real functional language (like references or modularity).

References

[1] L. Cardelli. An implementation of $f_{<:}$. Technical Report 97, SRC Research Report, 1993.

[2] P.-L. Curien and G. Ghelli. Subtyping + extensionality: confluence of $\beta\eta$-reduction in $f_{\leq}$. In *Theoretical Aspects of Computer Software*, volume 526, 1991. Lecture Notes in Computer Sciences.

[3] P. Giannini and S. Ronchi Dela Rocca. Type inference in polymorphic type discipline. In *Theoritical Aspects of Computer Software*. Springer Verlag, 1991. Lectures Notes in Computer Sciences.

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

[7] 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.

[8] J.-L. Krivine. *Lambda-Calculus : Types and Models*. Computers and their applications. Ellis Horwood, 1993.

[9] D. Leivant. Reasoning about functional programs and complexity classes associated with type disciplines. In *24th Annual Symposium on Foundations of Computer Science*, volume 44, pages 460–469, 1983.

[10] D. Leivant. Finitely stratified polymorphism. *Inf. Comput.*, pages 93–113, 1991.

[11] N. McCracken. The typechecking of programs with implicit type structure. In *Semantics of Data Types*, pages 301–315. Springer-Verlag, 1984. Lecture Notes in Computer Sciences.

[12] R. Milner. The standard ML core language. Polymorphism, 1985.

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

[14] Piperno and Ronchi della Rocca. type reconstruction in f using eta expansions. In *Logic in Computer Science*, 1994.

[15] 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.

[16] M Steffen and B Pierce. Higher-order subtyping. Technical Report ECS-LFCS-94-280, LFCS, University of Edinburgh, 1994.

[17] J. Tiuryn and P. Urzyczyn. The subtyping problem for second-order types is undecidable. 1995.

[18] 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.