System ST
Toward A Type System for Extraction AND Proofs of Programs

Christophe Raffalli

Laboratoire de Mathématique, Université de Savoie, 73376 Le Bourget-du-Lac cedex (FRANCE)

Abstract

We introduce a new type system called "System ST" (ST stands for SubTyping), based on subtyping, and prove the basic property of the system. We show the extraordinary expressive power of the system which leads us to think that it could be a good candidate for doing both proof and extraction of programs.

Key words: lambda-calcul, type, subtype MSC codes: 03B15, 03B40, 68N18

1 Introduction.

The Curry-Howard [1] isomorphism establishes a relation between proofs and programs. It has been extensively used to develop *type systems* (for instance COQ [2], AF2 [3]) where it is possible to extract a program from a proof of the existence of the function computed by the program. However, there are programs that can not be extracted from a proof even when we can prove that these programs are correct [4]. System ST is an attempt to enlarge the set of programs that can be extracted from a proof.

However, the initial goal of this work was not this one. It was to develop a type system, with nice properties, allowing the removal of parts of proofs with no algorithmic content when extracting programs. Works in this setting already exists [5,6], but they start from a proof, for instance in Girard's system F [7], and try to discover the computationally useless parts of this proof.

There is one drawback to this approach: it is not compatible with "separate compilation": The same lemma/subprogram needs to be recompiled every time

Email address: `Christophe.Raffalli@univ-savoie.fr` (Christophe Raffalli).
URL: `http://www.lama.univ-savoie.fr/ raffalli` (Christophe Raffalli).

it is used to compute the needed algorithmic content. Moreover, in practice, it is often the same part of a lemma that have algorithmic content (for instance, on natural numbers, the ordering relations have usually no algorithmic content).

Here we tried to develop a type system that was intended to be used directly to build proofs, using a type system where the parts of the proof that have algorithmic content can be identified in a nice way (This does not forbid to try to optimize programs later using algorithm similar to those developed in [5,6]). This has the following consequences:

- The system has to be elegant to be directly usable.
- We directly build a proof where the computationally useless parts are identified and there is no need to care about an algorithm to discover them.
- Therefore, we are free to design a powerful system with no care about the decidability of some particular problems.

The key idea of our system is to have two kinds of formulae: with and without algorithmic content. This idea is already present in the work of Christine Paulin-Mohring for CoQ [8] and also in the work of Nora Szasz and Paula Severi for Martin-Löf's Type Theory [9].

In our system, both kinds of formulae have an implication and a universal quantification and they interact through a subtyping predicate $A \subset B$ and a special implication $P \Rrightarrow A$. $A \subset B$ builds a formula with no algorithmic content from two formulae A and B with algorithmic content and means that any program that inhabits type A also inhabits type B. $P \Rrightarrow A$ builds a formula with algorithmic content from a formula P with no algorithmic content and a formula A with algorithmic content. The meaning of this new implication is the meaning of A if P is true and the type containing all λ-terms otherwise.

Then, with this simple idea, using very natural rules, we were very surprised by the expressive power of the system. This paper, after presenting system ST and its semantics in section 2 and 3, mainly studies its expressive power and the resulting properties:

- All the operators you can imagine are definable in system ST (like union types, intersection types, etc.). This defined operators have the desired properties, sometimes by adding an axiom. See section 4.
- Even singleton types can be defined. Singleton types are types only inhabited by one λ-term. As a consequence, we can prove that the system has the subject reduction property for η and the subject expansion for β. See section 5. Subject reduction for β will be discussed in a specific section **??** because it leads to interesting questions and problems.
- Surprisingly complex examples can be achieved in our system: we will show that we can derive Parigot's TTR rules for fixpoint operator [10] (one rule is based on well founded relation, the other is based on a logical fixpoint

operator). This is really surprising, because the only proof known from the author that the rule for the fixpoint operator is sound uses ordinal induction! Moreover, the rules of the system made the proof quite easy to find!! See section 6.

We will also show that theorems on data-types can be proved internally in the system (in some sense). We will illustrate this with a proof that the type of Church numerals is only inhabited by Church numerals.

The two last points give some strong indications that our system can be used both for proving and extracting programs (the system can prove properties of fixpoint operator and data-types!). We will only briefly introduce this topic in the concluding section of the paper about "work in progress".

Other work on subtyping: There are many papers and books using subtyping, mainly in the framework of type systems for programming languages (see [11,12] for a long but still incomplete list of references). However, our work is original because of the bidirectional interaction between typing and subtyping. In all the papers known from the author, we can only deduce a typing judgment from a subtyping judgment. In our system we can do it in both direction! In System ST, typing and subtyping judgments have similar roles.

Other work on intersection, union and singleton types: This kind of types have already been studied, for instance in [13,14,15,16,17]. Intersection types were introduced for theoretical reason in system D, and union type and singleton types where studied because they could be useful to design a module system for typed programming languages. System ST seems to be the "least common multiple" of all these systems and is indeed more powerful, for at least two reasons:

- These types are defined from more primitive ones.
- The four examples given in section 6 do not seem to be derivable (and often they can not be stated) in these previous systems (although the author did not check all the possible combinations). One of the most important case are the two fixpoint rules of Parigot's TTR type system [10] which are derivable in system ST: we can make use of the fact that the system ST in not normalizable to give meaningful types to non normalizable terms without adding rule or axiom about fixpoint combinator in the system.

PhoX: All the proofs in System ST (not the semantical proofs) have been realized and checked with the PhoX proof assistant [18].

Thanks to the anonymous referee, Ralph Matthes and Fréderic Ruyer for their comments that helped in improving the paper.

2 Definitions of the system.

2.1 Syntax.

We chose to formalize our system in a higher-order setting [19]. There are two reasons: this is the best we can do and the presentation is more concise than a second order version. Moreover, we need at least second order for some of our examples. However, most of the paper can be read with propositional logic in mind.

Definition 1 (sorts) *A sort is either a basic sort or a function sort between two sorts of smaller size (denoted $s \to s'$). Among the basic sorts, there are at least the sorts τ and o.*

The sort o will be the sort of formulae without algorithmic content (we will call them propositions. Church used omicron in [19]) and the sort τ is the sort of formulae with algorithmic content (we will call them types).

Definition 2 (expressions) *The set of expressions is the set of simply typed λ-terms, using sorts as simple types, written using the following constants of given sort:*

- $\Rightarrow_o : o \to o \to o$
- $\Rightarrow_\tau : \tau \to \tau \to \tau$
- $\Rightarrow\!\!\!\!\Rightarrow : o \to \tau \to \tau$
- $\subset : \tau \to \tau \to o$
- $\forall^s_o : (s \to o) \to o$ *for any sort* s.
- $\forall^s_\tau : (s \to \tau) \to \tau$ *for any sort* s.

We consider β-equivalent expressions to be equal.

Remark: later we will use pure λ-calculus. The expressions introduced above are the logical part of the system and despite the fact these are simply typed λ-term, they are distinct from the pure λ-terms that will be associated with proofs. In short, an object on the left of the ":" sign will be a pure λ-terms and an object on the right will be an expression.

Notation issues: We assume that λ-variables always carry their sort. We will write them x^s when the context do not impose the sort of the λ-variable. Otherwise, we will write only x when the sort of x is clear. In the same spirit, we will write $\lambda x\, t$ for $\lambda x^s\, t$ when the context makes it clear that s is the sort of x.

We will sometimes write applications using the standard mathematical notation: $f(x)$ for fx and $f(x,y)$ for fxy.

To make it easier to read formulae, we adopt the convention that the letters $A, B, C, \ldots, H$ represent types of the sort τ, that is formulae with algorithmic content and the letters $P, Q, R, \ldots, W$ represent propositions of the sort o that is formulae without algorithmic content.

To simplify writing, we will often write $\forall$ for $\forall^s_\chi$ and $\Rightarrow$ for $\Rightarrow_\chi$ for any value of χ or s. The above convention will always make it possible to recover the missing information. We will also write $\forall x\, A$ instead of $\forall(\lambda x\, A)$ and we will use the symbol $\Rightarrow$, $\subset$ and $\Rrightarrow$ in infix notation (this means we write $A \subset B$ instead of $\subset A\, B$).

To limit the number of parenthesis, we will consider that our implications are right associative with equal priorities, that subtyping has the same priority as implication (thus parenthesis are needed) and that quantification has the highest priority: $A \Rightarrow B \Rightarrow C$ means $A \Rightarrow (B \Rightarrow C)$, $A \Rightarrow P \Rrightarrow C$ means $A \Rightarrow (P \Rrightarrow C)$ and $\forall x\, A(x) \subset B$ means $(\forall x\, A(x)) \subset B$.

Definition 3 (context) *A context is a set composed of propositions P of sort o and pairs of the form $x : A$, where A is a type of sort τ and x is a λ-variable. Moreover, each λ-variable must be declared at most once in a context.*

Example 4 *The set $x : A, y : B, P, Q, R$ is a context if A and B have the sort τ and P, Q, R have the sort o*

Definition 5 (sequent) *There are two kinds of sequents: sequents of the form $\Gamma \vdash t : A$ and $\Gamma \vdash P$ where Γ is a context, t is a λ-term, A is a type and P is a proposition.*

2.2 Rules.

Axiom rules :

$$\frac{}{\Gamma, x : A \vdash x : A} \qquad \frac{}{\Gamma, P \vdash P}$$

Subtyping rules :

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

Implication rules (with algorithmic content):

$$\frac{\Gamma, x : A \vdash t : B}{\Gamma \vdash \lambda x\, t : A \Rightarrow B} \qquad \frac{\Gamma \vdash t : A \Rightarrow B \quad \Gamma \vdash u : A}{\Gamma \vdash (t\, u) : B}$$

$$\frac{\Gamma \vdash B \subset A \quad \Gamma \vdash A' \subset B'}{\Gamma \vdash (A \Rightarrow A') \subset (B \Rightarrow B')}$$

The subtyping rule for the implication will be called *contraposition*.

Implication rules (without algorithmic content):

$$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \Rightarrow Q} \qquad \frac{\Gamma \vdash P \Rightarrow Q \quad \Gamma \vdash P}{\Gamma \vdash Q}$$

Quantification rules (with algorithmic content):

$$\frac{\Gamma \vdash t : A(y)}{\Gamma \vdash t : \forall x\ A(x)}\ \dagger \qquad \frac{\Gamma \vdash t : \forall x\ A(x)}{\Gamma \vdash t : A(v)}\ \ddagger$$

$$\frac{\Gamma \vdash A \subset B(y)}{\Gamma \vdash A \subset \forall x\ B(x)}\ \dagger \qquad \frac{\Gamma \vdash A(v) \subset B}{\Gamma \vdash \forall x\ A(x) \subset B}\ \ddagger$$

Quantification rules (without algorithmic content) :

$$\frac{\Gamma \vdash P(y)}{\Gamma \vdash \forall x\ P(x)}\ \dagger \qquad \frac{\Gamma \vdash \forall x\ P(x)}{\Gamma \vdash P(v)}\ \ddagger$$

$\dagger$: y is a variable having the same sort than the variable x and not free in the conclusion of the rule.
$\ddagger$: v is an expression having the same sort than the variable x.

Special implication rules:

$$\frac{\Gamma, P \vdash t : A}{\Gamma \vdash t : P \twoheadrightarrow A} \qquad \frac{\Gamma \vdash t : P \twoheadrightarrow A \quad \Gamma \vdash P}{\Gamma \vdash t : A}$$

$$\frac{\Gamma, P \vdash A \subset B}{\Gamma \vdash A \subset (P \twoheadrightarrow B)} \qquad \frac{\Gamma \vdash A \subset B \quad \Gamma \vdash P}{\Gamma \vdash (P \twoheadrightarrow A) \subset B}$$

Mitchell's like axioms: We add immediately two axioms: the first one is Mitchell's axiom [20] for quantification. It is needed to give the completeness of subtyping. The second one is a similar axiom concerning our special implication.

Axiom 6

$$\vdash \forall A, B\ (\forall x\ (A(x) \Rightarrow B(x)) \subset (\forall x\ A(x) \Rightarrow \forall x\ B(x)))$$

$$\vdash \forall P\ \forall A, B\ ((P \twoheadrightarrow A \Rightarrow B) \subset (A \Rightarrow P \twoheadrightarrow B))$$

Fact 7 *We prove immediately*

$$\vdash \forall A\ \forall B\ (\forall x\ (A(x) \Rightarrow B) \subset (\forall x\ A(x) \Rightarrow B))$$
$$\vdash \forall A\ \forall B\ (\forall x\ (A \Rightarrow B(x)) \subset (A \Rightarrow \forall x\ B(x)))$$
$$\vdash \forall P\ \forall A\ (\forall x\ (P(x) \twoheadrightarrow A(x)) \subset (\forall x\ P(x) \twoheadrightarrow \forall x\ A(x)))$$
$$\vdash \forall P\ \forall A\ (\forall x\ (P(x) \twoheadrightarrow A) \subset (\forall x\ P(x) \twoheadrightarrow A))$$
$$\vdash \forall P\ \forall A\ (\forall x\ (P \twoheadrightarrow A(x)) \subset (P \twoheadrightarrow \forall x\ A(x)))$$

Proof: Immediate using the subtyping rules. We only use the first permutation axiom to prove the two first properties. ∎

The axiom 6 and fact 7 will be referred as *permutations* or *inversions*.

Other axioms: We will add other axioms to the system, but we need to give some definitions first and we prefer to add an axiom when it becomes useful. To help the reader, we give in appendix A all the axioms and definitions of system ST.

3 Semantics.

When we add more axioms, to make sure we introduce no contradiction, we use a set theoretic semantics.

We first give some notations: we denote by Λ the set of all untyped λ-term. We will only consider set of λ-terms closed for $\beta\eta$-equivalence. Therefore, when we define a set of λ-terms, this property is always implicit. For instance, $\{\lambda x\, x\}$ denotes the set of all λ-terms $\beta\eta$-equivalent to $\lambda x\, x$. However, we do not consider pure λ-terms that are equivalent to be equal (as we do for expressions).

Definition 8 (interpretation domain) *For each sort s we give an interpretation domain $\overline{s}$ as follows:*

- $\overline{\tau} = \mathcal{P}_{\beta\eta}(\Lambda)$*, the set of all sets of pure λ-terms closed for the $\beta\eta$-equivalence.*
- $\overline{o} = \{0, 1\} = \{\emptyset, \Lambda\}$*. The set notation for 0 and 1 will allow a more uniform presentation of the semantics.*
- $\overline{s \to s'}$ *is the set of all total functions from $\overline{s}$ to $\overline{s'}$*
- $\overline{s}$ *can be any set if s is a basic sort distinct from o and τ.*

Definition 9 (interpretation) *An interpretation $\mathcal{I}$ is a mapping associating to a variable x^s of sort s an element $\mathcal{I}(x)$ of $\overline{s}$.*

As usual, we write $\mathcal{I}[x_1 = v_1, \ldots, x_n = v_n]$ *to denote the interpretation where we changed only the value of the variables* $x_1, \ldots, x_n$.

We define by induction the interpretation $|t|^{\mathcal{I}}$ *as an element of* $\overline{s}$ *if* t *is an expression of sort* s*:*

- $|x|^{\mathcal{I}} = \mathcal{I}(x)$.
- $|t\,u|^{\mathcal{I}} = |t|^{\mathcal{I}}(|u|^{\mathcal{I}})$.
- $|\lambda x^s\, t|^{\mathcal{I}} = \lambda a\, |t|^{\mathcal{I}[x^s = a]}$.
- $|\Rightarrow_s|^{\mathcal{I}} = \lambda a \lambda b\, \{t \mid \forall u \in a, (t\,u) \in b\}$ *for* $s = o$ *or* $s = \tau$ *(for* $s = o$*, it is easy to check that we get classical semantics).*
- $|\Rrightarrow|^{\mathcal{I}} = \lambda a \lambda b\, (\textit{if } a = 1 \textit{ then } b \textit{ else } \Lambda)$ *where* Λ *is the set of all* λ*-terms.*
- $|\subset|^{\mathcal{I}} = \lambda a \lambda b\, (\textit{if } a \subset b \textit{ then } 1 \textit{ else } 0)$.
- $|\forall^s_{s'}|^{\mathcal{I}} = \lambda a \bigcap_{b \in \overline{s}} a(b)$ *for* $s' = o$ *or* $s' = \tau$ *(universal quantification is interpreted by an intersection over the interpretation domain of the quantified sort).*

Definition 10 *We say that an interpretation* $\mathcal{I}$ *and a* λ*-substitution* σ *satisfy a context* $\Gamma = x_1 : A_1, \ldots x_n : A_n, P_1, \ldots, P_q$ *if* $x_i\sigma \in |A_i|^{\mathcal{I}}$ *for* i *in* $\{1, \ldots, n\}$ *and* $|P_j|^{\mathcal{I}} = 1$ *for* j *in* $\{1, \ldots, q\}$.

We write this $\mathcal{I}, \sigma \models \Gamma$.

Theorem 11 (correctness) *For any interpretation* $\mathcal{I}$*, any substitution* σ *and any context* Γ *such that* $\mathcal{I}, \sigma \models \Gamma$ *we have*

$$\Gamma \vdash t : A \textit{ implies } t\sigma \in |A|^{\mathcal{I}} \textit{ and } \Gamma \vdash P \textit{ implies } |P|^{\mathcal{I}} = 1$$

Proof: By induction on the size of proofs using the definition of the semantics. The correctness will be preserved by all the rules of system ST as given in appendix (page 25).

Remark: we only use the fact that sets of λ-terms are closed for β-expansion (for the implication introduction rule) and η-reduction (for the permutation axioms). Thus we could generalize our semantics using sets of λ-terms closed only for these relations. ∎

Important remark: The interpretation of $\forall K\, (P(K) \Rightarrow K)$ is the intersection of all sets K having the property P. We use this construction very often (for instance below for the definition of $\top_o$).

4 First development.

4.1 About false and truth.

Definition 12 *Having two kinds of formulae, we can define two kinds of false and truth:*

- $\bot_\tau := \forall^\tau_\tau X\, X$
- $\bot_o := \forall^\tau_o X\, \forall^\tau_o Y\, (X \subset Y)$
- $\top_\tau := \forall^\tau_\tau K (\forall^\tau_o X (X \subset K) \Rrightarrow K)$
- $\top_o := \forall^\tau_o X\, (X \subset X)$

Remark: other definitions were possible, for instance $\bot_o := \forall^o_o X\, X$. The choices were made to have the next property and not to use variables of sort o (it seems we never need such variables).

Fact 13 *The interpretation of $\bot_\tau, \bot_o$ and $\top_\tau, \top_o$ are respectively the minimal and maximal element in $\overline{\tau}$ and $\overline{o}$. This means that $|\bot_\tau| = \emptyset$, $|\bot_o| = 0$, $|\top_\tau| = \Lambda$, and $|\top_o| = 1$.*

Proof: The interpretations of $\bot_o, \bot_\tau$ and $\top_o$ are clear. From the definition, we see that the interpretation of $\top_\tau$ is the intersection of all sets of λ-terms K containing all sets of λ-term X. This intersection only ranges over $K = \Lambda$. ∎

Axiom 14 *We add to the system one axiom and two rules useful to reason with $\bot_\tau$:*

$$\forall A,B\, (A \subset (\bot_\tau \Rightarrow B)) \qquad \textit{emptiness}$$

$$\frac{\Gamma \vdash t : (P \Rrightarrow \bot_\tau) \Rightarrow \bot_\tau}{\Gamma \vdash P} \quad \textit{classical reasoning}$$

$$\frac{\Gamma, x : A \vdash A \subset B}{\Gamma \vdash A \subset B}$$

Remark: the axiom means that any set X is a subset of $\bot_\tau \Rightarrow Y$. This enforces that $\bot_\tau$ is interpreted by the empty set (there may be other semantics where this axiom is true but the author doesn't know any, because if $\bot_\tau$ is non empty, $\bot_\tau \Rightarrow \bot_\tau$ is in general a strict subset of $\top_\tau$).

Remark: the first rule is a strange form of classical reasoning. The idea is that we want to have classical reasoning for proposition of sort o because they are interpreted classically. The natural axiom would be $\forall P\, (((P \Rightarrow \bot_o) \Rightarrow \bot_o) \Rightarrow P)$. Our rule is both stronger (see fact 17) and necessary because no rule except this one can deduce a sequent with no algorithmic content from a sequent with some algorithmic content.

Remark: the last rule seems not related to false. But, as we see in the correctness proof below, it is justified by classical case splitting over the clause "the type A is inhabited".

Fact 15 *These axiom and rules are correct in our semantics.*

Proof: For the axiom, this is immediate. For the first rule, if P is interpreted by 0 then $(P \Rrightarrow \bot_\tau) \Rightarrow \bot_\tau$ is interpreted by the empty set and this means that the sequent Γ is unsatisfiable and if P is interpreted by 1, the conclusion sequent is true.

For the last rule, if A is interpreted by the empty set, then the conclusion is true. If A is not interpreted by the empty set, then, Γ satisfiable implies $\Gamma, x : A$ satisfiable using the same interpretation. ∎

Fact 16 *Both definitions of False are equivalent. The implication in one direction can be expressed by the formula $\bot_o \Rrightarrow \bot_\tau$. The converse implication can only be written as a rule because we did not introduce an implication of sort $\tau \to o \to o$. This derived rule is:*

$$\frac{\Gamma \vdash t : \bot_\tau}{\Gamma \vdash \bot_o}$$

Proof: This rule is an immediate consequence of our classical reasoning rule. The converse implication is easy by deducing $\forall X\ (X \Rightarrow X) \subset \bot_\tau$ from $\bot_o$ and therefore $\lambda x\, x : \bot_\tau$. ∎

Fact 17 *We can prove $\forall P\ (((P \Rightarrow \bot_o) \Rightarrow \bot_o) \Rightarrow P)$.*

Proof: We assume $(P \Rightarrow \bot_o) \Rightarrow \bot_o$ (1) and prove P. Using our absurdity rule, we need to prove t : $(P \Rrightarrow \bot_\tau) \Rightarrow \bot_\tau$ for some t which is immediate by (1) and fact 16. ∎

Corollary 18 *We can deduce the following rules of excluded middle (the first rule allows to distinguish the cases were P are true and false, the second one allows to distinguish the cases where A is inhabited or empty):*

$$\frac{\Gamma, P \vdash Q \quad \Gamma, P \Rightarrow \bot_o \vdash Q}{\Gamma \vdash Q} \qquad \frac{\Gamma, x : A \vdash Q \quad \Gamma, A \subset \bot_\tau \vdash Q}{\Gamma \vdash Q}$$

Proof: The first one is (as usual) an immediate consequence of fact 17. For the second one, we prove Q using fact 17: we assume $Q \Rightarrow \bot_o$ and prove $\bot_o$. Then, using the right premise, it is enough to prove $A \subset \bot_\tau$ and using the last rule in axiom 14, we can assume $x : A$. Then, we deduce Q by the second premise. Therefore, we have $\bot_o$ from which we deduce $A \subset \bot_\tau$. ∎

As a consequence, we can prove the following lemma that we will use later:

Lemma 19 (Modified contraposition) *We can derive the following modified rule for contraposition:*

$$\frac{\Gamma \vdash B \subset A \quad \Gamma, x : B \vdash A' \subset B'}{\Gamma \vdash (A \Rightarrow A') \subset (B \Rightarrow B')}$$

Proof: It is an immediate consequence of the previous fact, distinguishing the cases where B is empty or not. The premises solves the case where B is inhabited. In the other case we conclude using the axiom in 14. ∎

4.2 Restriction types.

Definition 20 *We define Parigot's [10] restriction operator of system TTR as follows:*

$$A \upharpoonright P := \forall K\, ((A \subset (P \Rightarrow\!\!\!\!\Rightarrow K)) \Rightarrow\!\!\!\!\Rightarrow K)$$

The meaning of $A \upharpoonright P$ is A if P is true and $\bot_o$ otherwise. $A \upharpoonright P$ can also be seen as a conjunction where we keep only the algorithmic content of A.

Remark: we could also define $A \upharpoonright P := \forall K\, ((P \Rightarrow A \subset K) \Rightarrow\!\!\!\!\Rightarrow K)$ but it is very easy to prove $\forall P\, \forall A,B\, ((P \Rightarrow A \subset B) \Leftrightarrow (A \subset (P \Rightarrow\!\!\!\!\Rightarrow B)))$ (1) which shows that both definitions are trivially equivalent. Equivalence (1) suggests that the symbol $\Rightarrow_o$ may not be necessary, but it is very convenient to use it in some case (for instance to define equivalence and be able to write the proposition (1) above). In fact, if we have no variable with a sort ending by o, all formulae are equivalent to a formula without the symbol $\Rightarrow_o$.

Fact 21 *For any interpretation $\mathcal{I}$, $|A \upharpoonright P|^{\mathcal{I}}$ is interpreted by $|A|^{\mathcal{I}}$ if $|P|^{\mathcal{I}} = 1$ and by $\emptyset$ if $|P|^{\mathcal{I}} = 0$.*

Proof: Easy by definition of the interpretation. ∎

Fact 22 *We can derive the following rules for the restriction operator:*

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash P}{\Gamma \vdash t : A \upharpoonright P} \qquad \frac{\Gamma \vdash t : A \upharpoonright P}{\Gamma \vdash t : A} \qquad \frac{\Gamma \vdash t : A \upharpoonright P}{\Gamma \vdash P}$$

$$\frac{\Gamma \vdash A \subset B \quad \Gamma \vdash P}{\Gamma \vdash A \subset (B \upharpoonright P)} \qquad \frac{\Gamma, P \vdash A \subset B}{\Gamma \vdash (A \upharpoonright P) \subset B}$$

Proof: All rules are easy except the third one which deduce a sequent without algorithmic content from a sequent with algorithmic content. We deduce it from our classical rule and the fact 16. ∎

Fact 23 *We can prove an inversion statement (analogous to the axioms 6) for the restriction operator:*

$$\forall P\, \forall A,B\, ((P \Rrightarrow A \Rightarrow B) \subset ((A \upharpoonright P) \Rightarrow B)).$$

Proof: From the fact 17, we can distinguish the case when P is true and the case when $P \Rightarrow \bot_o$ is true. In both cases the proof is easy. ∎

4.3 Union types.

Definition 24 (Union types) *The following type defines the* union *of all $A(x)$:*

$$\bigcup x\, A(x) := \forall K\, (\forall x\, (A(x) \subset K) \Rrightarrow K)$$

Fact 25 *For any interpretation $\mathcal{I}$ and any formula $A(x)$ where x is of sort s, $|\bigcup x\, A(x)|^{\mathcal{I}} = \bigcup_{\phi \in \overline{s}} |A(x)|^{\mathcal{I}[x=\phi]}$.*

Proof: immediate using the definition of the semantics.

Remark: the previous proof is not possible if we choose for $\overline{\tau}$ a set of sets of λ-terms only closed for implications and arbitrary intersections and not closed for arbitrary unions. This justifies the need for the following axiom which is not true in such a weak semantics:

Axiom 26 (Union axiom)

$$\forall F\, \forall A\, (\forall x\, (F(x) \Rightarrow A) \subset (\bigcup x\, F(x) \Rightarrow A))$$

Fact 27 *This axiom is true in our semantics.*

Proof: Easy using the fact 25 expressing that our union operator is really interpreted by a union. ∎

Fact 28 (Union rules) *We can derive the following rules for union:*

$$\frac{\Gamma \vdash t : F(v)}{\Gamma \vdash t : \bigcup x\, F(x)}\ \ddagger \qquad \frac{\Gamma \vdash t : \bigcup x\, F(x) \quad \Gamma \vdash u : F(y) \Rightarrow A}{\Gamma \vdash (u\, t) : A}\ \dagger$$

$$\frac{\Gamma \vdash A \subset F(v)}{\Gamma \vdash A \subset \bigcup x\, F(x)}\ \ddagger \qquad \frac{\Gamma \vdash F(y) \subset A}{\Gamma \vdash \bigcup x\, F(x) \subset A}\ \dagger$$

Proof: Easy, using the union axiom only for the second rule. ∎

Important remark: The interpretation of $\bigcup K\, (K \upharpoonright P(K))$ is the union of all sets K having the property P. We use this construction very often (for instance

below for the definition of the inverse image).

4.4 Inverse and direct images.

Definition 29 *We can define the direct and inverse images as follows:*

$$A[B] := \forall X\,((A \subset (B \Rightarrow X)) \twoheadrightarrow X)$$
$$A^{-1}[B] := \bigcup X\,(X \upharpoonright (A \subset (X \Rightarrow B)))$$

Fact 30 *The interpretation of the direct and inverse images are:*

$$|A[B]|^{\mathcal{I}} = \{(t\,u) \mid t \in |A|^{\mathcal{I}}, u \in |B|^{\mathcal{I}}\}$$
$$|A^{-1}[B]|^{\mathcal{I}} = \{u \mid \forall t \in |A|^{\mathcal{I}}, (t\,u) \in |B|^{\mathcal{I}}\}$$

Proof:

- $A[B]$: The right-left inclusion is easy. For the left-right inclusion, we denote $\Phi = \{(t\,u) \mid t \in |A|^{\mathcal{I}}, u \in |B|^{\mathcal{I}}\}$. If w belongs to the interpretation of $A[B]$, then $w \in |(A \subset (B \Rightarrow X)) \twoheadrightarrow X|^{\mathcal{I}[X=\Phi]}$ then it is easy to check that $|A \subset (B \Rightarrow X)|^{\mathcal{I}[X=\Phi]} = 1$ which implies $w \in \Phi$.
- $A^{-1}[B]$: For the right-left inclusion, we take u such that $\forall t \in |A|^{\mathcal{I}}, (t\,u) \in |B|^{\mathcal{I}}$. Therefore, we have $|A \subset (X \Rightarrow B)|^{\mathcal{I}[X:=\{u\}]} = 1$. From this we deduce that $u \in |A^{-1}[B]|^{\mathcal{I}}$.

  For the left-right inclusion, we take $u \in |A^{-1}[B]|^{\mathcal{I}}$ and $t \in |A|^{\mathcal{I}}$. By fact 25 we find $\Phi \in \overline{\tau}$ such that $u \in |X \upharpoonright (A \subset (X \Rightarrow B))|^{\mathcal{I}[X=\Phi]}$. Therefore, we have $u \in \Phi$ and $|A \subset (X \Rightarrow B)|^{\mathcal{I}[X=\Phi]} = 1$ which implies $(t\,u) \in |B|^{\mathcal{I}}$. ∎

Fact 31 *We can derive the following rule and theorems for direct and inverse image in our system:*

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash u : B}{\Gamma \vdash (t\,u) : A[B]}$$

$$\vdash \forall A, B, C\,((A \subset (B \Rightarrow C)) \Rightarrow A[B] \subset C).$$
$$\vdash \forall A, B, C\,((A[B] \subset C) \Rightarrow A \subset (B \Rightarrow C)).$$
$$\vdash \forall A, B, C\,((A \subset (B \Rightarrow C)) \Rightarrow B \subset A^{-1}[C]).$$
$$\vdash \forall A, B, C\,((B \subset A^{-1}[C]) \Rightarrow A \subset (B \Rightarrow C)).$$

Proof: The rule and the first and third properties are easy to prove. We give details of the proof of the two other properties.

To prove the second property, we assume $A[B] \subset C$ (1) and we must prove $A \subset (B \Rightarrow C)$. from (1) we get $(B \Rightarrow A[B]) \subset (B \Rightarrow C)$. Therefore, using

transitivity, we just need to prove $A \subset (B \Rightarrow A[B])$. This means we must prove $A \subset (B \Rightarrow \forall X\,((A \subset (B \Rightarrow X)) \twoheadrightarrow X))$. Then, we easily prove $A \subset \forall X\,((A \subset (B \Rightarrow X)) \twoheadrightarrow B \Rightarrow X)$ and deduce the result from our inversion properties (fact 7) and the transitivity.

For the fourth property, we first prove $\forall A, C\,(A \subset (A^{-1}[C] \Rightarrow C))$ (1). This means we must prove $A \subset (\bigcup X\,(X \upharpoonright (A \subset (X \Rightarrow C))) \Rightarrow C)$. From the inversion property (fact 23) of the restriction operator, we get $A \subset \forall X\,(X \upharpoonright (A \subset (X \Rightarrow C)) \Rightarrow C)$ and we end the proof of (1) using the union axiom 26.

Then, we assume $B \subset A^{-1}[C]$ (2) and prove $A \subset (B \Rightarrow C)$. By (1) and transitivity, we just need to prove $(A^{-1}[C] \Rightarrow C) \subset (B \Rightarrow C)$ which is immediate from (2). ∎

4.5 Other operators.

Many other operators can be defined in the system. We just give three examples:

Definition 32 *We define*

- $A \cap B := \bigcup X\,(X \upharpoonright ((X \subset A) \wedge (X \subset B)))$
- $A \cup B := \forall K\,((A \subset K) \twoheadrightarrow (B \subset K) \twoheadrightarrow K)$
- $P \,||\, Q := \forall X\,((P \twoheadrightarrow X) \Rightarrow (Q \twoheadrightarrow X) \Rightarrow X)$

The conjunction, in the first definition, is $P \wedge Q := \forall K\,((P \Rightarrow Q \Rightarrow K) \Rightarrow K)$ *(we could also have defined* $A \cap B := \bigcup X\,((X \upharpoonright (X \subset A)) \upharpoonright (X \subset B))$*).*

It is easy to prove that:

- $A \cap B$ is interpreted by the intersection.
- $A \cup B$ is interpreted by the union.
- $P \,||\, Q$ is interpreted by a subset of $\{\lambda x \lambda y\, x, \lambda x \lambda y\, y\}$ containing the first term if and only if P is true and the second term if and only if Q is true.

All the natural rules for these new operators are easy to write and prove except the intersection rule of system D. We will prove in the next section that even this rule is admissible (which means it does not add new typing nor subtyping judgment). This is quite surprising because this rule contains a constraint on the λ-terms!

5 Toward subject reduction and expansion.

5.1 Singleton types.

The possibility of defining direct and inverse image in our system was a surprise for the author ! But, the following definition and facts are even more surprising ! For any pure lambda term t, we are going to define a type $|t|^\phi$, whose intended meaning is the singleton of t.

Definition 33 *We define $\Lambda X\, A(X) := \forall X\, (X \Rightarrow A(X))$. Then, for every λ-term t and every mapping ϕ from λ-variable to formula of sort τ, we define $|t|^\phi$ by induction as:*

- $|x|^\phi = \phi(x)$.
- $|(t\, u)|^\phi = |t|^\phi[|u|^\phi]$.
- $|\lambda x\, t|^\phi = \Lambda X |t|^{\phi[x=X]}$.

Fact 34 *For any interpretation $\mathcal{I}$ and any increasing predicate F of sort $\tau \to \tau$ (this means $\vdash \forall X, Y\, ((X \subset Y) \Rightarrow F(X) \subset F(Y))$), we have*

$$|\Lambda X\, F(X)|^{\mathcal{I}} = \{\lambda x\, t \mid \forall u \in \Lambda, t[x = u] \in |F(X)|^{\mathcal{I}[X=\{u\}]}\}$$

Proof: The left-right inclusion is immediate using an η-expansion. The converse uses the hypothesis that F is increasing. ∎

Corollary 35 *Let t be a λ-term, $\mathcal{I}$ an interpretation and ϕ a mapping from λ-variables to formulae such that for any variable x free in t, $|\phi(x)|^{\mathcal{I}}$ is a singleton (more precisely an equivalence class for the $\beta\eta$-equivalence), then the interpretation of $|t|^\phi$ is a singleton.*

Moreover, if t is a closed term, the interpretation of $|t|^\phi$, which is independent from ϕ or $\mathcal{I}$, is the singleton $\{t\}$.

Proof: If $\{x_1, \ldots, x_n\}$ are the free variables of t and if $|\phi(x_i)|^{\mathcal{I}} = \{u_i\}$, then we prove that the interpretation of $|t|^\phi$ is $\{t[x_1 = u_1, \ldots, x_n = u_n]\}$ by induction on the size of t using lemma 34. To be able to use this lemma we need first to prove that $\lambda X\, |t|^{\phi[x=X]}$ is increasing, which is an easy induction. ∎

Fact 36 *For any λ-term t, if $\{x_1, \ldots, x_n\}$ are the free variables of t, we can prove that $x_1 : \phi(x_1), \ldots, x_n : \phi(x_n) \vdash t : |t|^\phi$.*

Proof: The proof is an immediate induction on the size of t, using facts 31 and the definition of $\Lambda X\, F(X)$. ∎

This fact and lemma 35 mean that singleton types are definable in our system, using for ϕ an injective mapping from λ-variables to variables of sort τ!

Corollary 37 *System ST is not normalizable.*

Proof: We define the following formulae of sort τ:

$$\Delta_F := \Lambda X\ F[X[X]] \text{ and } \Psi := \Lambda F\ \Delta_F[\Delta_F].$$

We remark that $\Psi = |\lambda f((\lambda x(f(xx)))(\lambda x(f(xx))))|^\phi$. Therefore we have $\vdash Y : \Psi$ where $Y = \lambda f((\lambda x(f(xx)))(\lambda x(f(xx))))$ and Y is a fixpoint combinator and is not normalizable. ∎

Theorem 38 *Let Γ be $x_1 : A_1, \ldots, x_n : A_n, P_1, \ldots, P_q$ and ϕ defined by $\phi(x_i) = A_i$ for $i \in \{1, \ldots, n\}$. We can derive $\Gamma \vdash t : A$ if and only if we can derive $\Gamma \vdash |t|^\phi \subset A$.*

Proof: The right to left implication is a consequence of the previous result. The converse implication is proved by induction on the size of the derivation: each typing rule is mirrored by a subtyping rule acting on the right side of the $\subset$ sign except the implication introduction and elimination.

The first case is solved using the modified contraposition rule (lemma 19) and the second case using the first property in the fact 31.

We do not even need to consider rules whose conclusion are not a typing sequent. Therefore, we can add any axiom with no algorithmic content and preserve this result (there is still one more axiom to add).

∎

Corollary 39 (Intersection rule) *The following rule for the intersection is admissible in the system:*

$$\frac{\Gamma \vdash t : A \quad \Gamma \vdash t : B}{\Gamma \vdash t : A \cap B}$$

Proof: From the premises of the rule we get $\Gamma \vdash |t|^\phi \subset A$ and $\Gamma \vdash |t|^\phi \subset B$ by theorem 38. From this we easily deduce $\Gamma \vdash |t|^\phi \subset A \cap B$ and the result follows by theorem 38. ∎

The above rule is not derivable but only admissible (the proof of the premises need to be transformed).

Corollary 40 (How to prove subject reduction) *Let R be a relation on λ-terms. If for any ϕ and any terms t, t', $t\ R\ t'$ implies $\vdash |t'|^\phi \subset |t|^\phi$ then, we have the subject reduction property for R (that is $t\ R\ t'$ and $\Gamma \vdash t : A$ imply*

$\Gamma \vdash t' : A$).

Proof: Follows from fact 36 and theorem 38. ∎

5.2 The easy part.

Theorem 41 *Our system has the subject-reduction property for η and the subject-expansion property for β.*

Proof: For the η-case, using the corollary 40, it is enough (it is even much stronger) to prove $A \subset \Lambda X\ A[X]$ for any formula A of sort τ. We have $\Lambda X\ A[X] = \forall X\ (X \Rightarrow \forall Y\ ((A \subset (X \Rightarrow Y)) \twoheadrightarrow Y))$. Using inversion properties, we get $\forall X, Y\ ((A \subset (X \Rightarrow Y)) \twoheadrightarrow X \Rightarrow Y) \subset \Lambda X\ A[X]$ and $A \subset \forall X, Y\ ((A \subset (X \Rightarrow Y)) \twoheadrightarrow X \Rightarrow Y)$ follows trivially from the subtyping rules.

For the β-case, it is enough (again much stronger) to prove $(\Lambda X\ A(X))[B] \subset A(B)$ for any A and B which is easy using the subtyping rules. ∎

Fact 42 *Subject reduction for β for the identity term and the constant terms ($\lambda x\, t$ with x not free in t) are respectively given by the following axiom and rule which are semantically true (we do not consider these to be part of system ST):*

$$\forall A, B\ ((\forall X\ (X \Rightarrow X) \subset (A \Rightarrow B)) \Rightarrow A \subset B)$$

$$\frac{\Gamma \vdash t : C \quad \Gamma \vdash (A \Rightarrow B) \subset (C \Rightarrow D)}{\Gamma \vdash B \subset D}$$

Proof: It is easy to see that they are both true in our semantics. For the β-reduction for the identity, we have to prove $\forall A\ (A \subset (\Lambda X\ X)[A])$. To do this, after applying few rules, we assume $\forall X_0\ (X_0 \Rightarrow X_0) \subset (A \Rightarrow X)$ and must prove $A \subset X$. This is an immediate consequence of the axiom.

For constant terms, we assume $x : A$ and we must prove $B \subset (\Lambda X\ B)[A]$ (the beta-reduction for constant terms is not true if the argument A is the empty set). Here again, after a few rules, we assume $\forall X_0\ (X_0 \Rightarrow B) \subset (A \Rightarrow X)$ and must prove $B \subset X$. From this hypothesis, we get easily $(\top_\tau \Rightarrow B) \subset (A \Rightarrow X)$, and then $B \subset X$ follows from our new rule. ∎

6 Surprising examples.

6.1 An easy example: Cartesian product.

Definition 43 *We define conjunction and Cartesian product as follows:*

- $A \mathbin{\mathbb{\wedge}} B := \forall X\,((A \Rightarrow B \Rightarrow X) \Rightarrow X)$
- $A \times B := \Lambda X\,(X[A])[B]$

Fact 44 *We prove* $\forall A,B\,(A \mathbin{\mathbb{\wedge}} B \subset A \times B)$ *and* $\forall A,B\,(A \times B \subset A \mathbin{\mathbb{\wedge}} B)$.

Proof: By a mechanical application of the rules, using fact 31 at the end of the proof of $\forall A,B\,(A \mathbin{\mathbb{\wedge}} B \subset A \times B)$. ∎

Using the interpretation of $\times$, this means that terms inhabiting $A \mathbin{\mathbb{\wedge}} B$ have the form $\lambda x(x\,a\,b)$ where a inhabits A and b inhabits B [1]. The usual proof of this fact is easy, but here in system ST, it could clearly be done automatically!

We have now two ways to prove $\vdash \lambda c(c\,\lambda x\lambda y\,x) : \forall A,B\,(A \mathbin{\mathbb{\wedge}} B \Rightarrow A)$. First by using the natural proof of $\forall A,B\,(A \mathbin{\mathbb{\wedge}} B \Rightarrow A)$ and extracting the term. Second using facts 36 and 44 and proving the following:

Fact 45 $\vdash \Lambda C\; C[\Lambda X\,\Lambda Y\,X] \subset \forall A,B\,(A \times B \Rightarrow A)$.

Proof: We have to prove $\Lambda C\; C[\Lambda X\,\Lambda Y\,X] \subset (A \times B \Rightarrow A)$. Using fact 31 it is enough to prove $(\Lambda C\; C[\Lambda X\,\Lambda Y\,X])[A \times B] \subset A$. Then, we conclude using four times the inclusion corresponding to the β-expansion in theorem 41 (that is $\forall A\;\forall B\;((\Lambda X\;A(X))[B] \subset A(B))$). ∎

Two remarks are important here: the previous proof is a proof that the program is correct using some kind of symbolic evaluation. To do the symbolic evaluation, we have used the subtyping relation corresponding to the β-expansion. This inversion shows why subject-reduction is not really necessary: this subtyping would be useful only to prove programs using β-expansion during a symbolic evaluation!

[1] this is only true when A and B are only inhabited by closed terms. This is related to the fact that system F does not have a real product enjoying the surjective pairing. The subtyping we can prove means that $A \times B$ is exactly the same thing than $A \mathbin{\mathbb{\wedge}} B$. Thus, this new definition does not solve the problem of surjective pairing.

6.2 First TTR fixpoint rule.

Definition 46 *Well founded relation can be defined by:*

$$W_f(R) := \forall P\ (\forall a\ (\forall b\ (R(b,a) \Rightarrow P(b)) \Rightarrow P(a)) \Rightarrow \forall a\ P(a)).$$

Remark: $W_f(R)$ is a proposition of sort o if $R(a,b)$ is of sort o.

Theorem 47 *We can derive the following rule in our system:*

$$\frac{\Gamma \vdash t : \forall y\ (\forall x\ (R(x,y) \twoheadrightarrow A(x)) \Rightarrow A(y)) \quad \Gamma \vdash W_f(R)}{\Gamma \vdash (Y\ t) : \forall y\ A(y)}$$

This theorem means that if we can prove that a relation is well-founded, we can prove it with algorithmic content and the algorithmic content is a fixpoint combinator.

Proof: We use Ψ,Δ_F and Y defined in the proof of corollary 37. We assume $\mathrm{W}_f(R)$ and prove $\Psi \subset (\forall y\ (\forall x\ (R(x,y) \twoheadrightarrow A(x)) \Rightarrow A(y)) \Rightarrow \forall y\ A(y))$ (then, we conclude using corollary 37). Let F be $\forall y\ (\forall x\ (R(x,y) \twoheadrightarrow A(x)) \Rightarrow A(y))$. We have to prove $\Psi \subset (\mathrm{F} \Rightarrow \forall y\ A(y))$. Using inversion, we just need to prove $\Psi \subset (\mathrm{F} \Rightarrow A(y))$. Using three subtyping rules, we now need to prove $\Delta_\mathrm{F}[\Delta_\mathrm{F}] \subset A(y)$. We prove this by well founded induction on y: we assume $\forall b\ (R(b,a) \Rightarrow \Delta_\mathrm{F}[\Delta_\mathrm{F}] \subset A(b))$ and we prove $\Delta_\mathrm{F}[\Delta_\mathrm{F}] \subset A(a)$. This comes from $\Delta_\mathrm{F} \subset (\Delta_\mathrm{F} \Rightarrow A(a))$, which comes from $\mathrm{F}[\Delta_\mathrm{F}[\Delta_\mathrm{F}]] \subset A(a)$, which comes from $\mathrm{F} \subset (\Delta_\mathrm{F}[\Delta_\mathrm{F}] \Rightarrow A(a))$, which comes from $R(b,a) \Rightarrow \Delta_\mathrm{F}[\Delta_\mathrm{F}] \subset A(b)$, which is a consequence of the induction hypothesis. ∎

6.3 Second TTR fixpoint rule.

We will now show how we can define the TTR least fixpoint operator in the system and derive its rules. This is very surprising for the fixpoint rule (allowing to type terms using a fixpoint combinator), because to prove the correctness of this rule we usually use ordinal induction. This is why we included this example here despite the fact that notations and expressions are difficult to read.

First we give some definitions:

Definition 48 (fixpoint operator and increasing predicates)

Fixpoint:	$\mu_1 X\ F(X) := \lambda x \forall Y\ ((F(Y)\ \subset_1\ Y) \Rrightarrow\ Y(x))$
	$\mu_1 X\ F(X) : s \to \tau$ *if* $F : (s \to \tau) \to (s \to \tau)$.
Pointwise subtyping	$F \subset_1\ G := \forall x\ (F(x) \subset\ G(x))$
	$F \subset_1\ G : o$ *if* $F, G : s \to \tau$.
Pointwise implication:	$F \Rightarrow_1\ G := \forall x\ (F(x) \Rightarrow\ G(x))$
	$F \Rightarrow_1\ G : \tau$ *if* $F, G : s \to \tau$.
F *is increasing:*	$Incr(F) := \forall X, Y\ ((X \subset_1\ Y) \Rightarrow (F(X) \subset_1\ F(Y)))$
	$Incr(F) : o$ *if* $F : (s \to \tau) \to (s \to \tau)$.

Fact 49 *The fixpoint operator is indeed a fixpoint for increasing predicate F. This means we can prove:*

$$\forall F\ (Incr(F) \Rightarrow\ (F(\mu_1 X\ F(X)) \subset_1\ \mu_1 X\ F(X)))$$
$$\forall F\ (Incr(F) \Rightarrow\ (\mu_1 X\ F(X) \subset_1\ F(\mu_1 X\ F(X))))$$

Proof: For the first property, we assume $Incr(F)$ (1), $F(X) \subset_1 X$ (2) and we prove $F(\mu_1 X_0\ F(X_0), x) \subset X(x)$. This comes from (2), the transitivity and $F(\mu_1 X_0\ F(X_0), x) \subset F(X, x)$. Using (1), this comes from $\mu_1 X_0\ F(X_0) \subset_1 X$ and is easily proved using the subtyping rules.

For the second property, we assume $Incr(F)$ (1) and we prove $(\mu_1 X\ F(X))(x) \subset F(\mu_1 X\ F(X), x)$. To get this, we prove $F(F(\mu_1 X\ F(X))) \subset_1 F(\mu_1 X\ F(X))$ which comes from (1) and the previous property. ∎

Lemma 50 *We prove the following lemma:*

$$\vdash \forall A\ \forall B\ \forall F \begin{pmatrix} \forall X\ ((A \subset (X \Rightarrow_1 B)) \Rightarrow (A \subset (F(X) \Rightarrow_1 B))) \Rightarrow \\ (A \subset (\mu_1 X\ F(X) \Rightarrow_1 B)) \end{pmatrix}.$$

Proof: We assume

$$\forall X\ ((A \subset (X \Rightarrow_1 B)) \Rightarrow A \subset (F(X) \Rightarrow_1 B))(1)$$

and we have to prove $A \subset ((\mu_1 X\ F(X))(x) \Rightarrow B(x))$. Using fact 31 it is enough to prove $(\mu_1 X\ F(X))(x) \subset A^{-1}[B(x)]$. This comes from $F(\lambda x_0(A^{-1}[B(x_0)])) \subset_1 \lambda x_0(A^{-1}[B(x_0)])$, which comes from $A \subset (F(\lambda x_1(A^{-1}[B(x_1)]), x_0) \Rightarrow B(x_0))$ (2) using again fact 31. Finally, we get (2) from (1) and $A \subset (\lambda x_1(A^{-1}[B(x_1)]) \Rightarrow_1 B)$, which is proved using fact 31 a third time. ∎

Lemma 51 *We prove (Ψ is defined in the proof of corollary 37):*

$$\vdash \forall F\, \forall K\, (\Psi \subset (\forall X\, ((X \Rightarrow_1 K) \Rightarrow (F(X) \Rightarrow_1 K)) \Rightarrow (\mu_1 X\, F(X) \Rightarrow_1 K))).$$

Proof: We define $\mathrm{H} = \forall X\, ((X \Rightarrow_1 K) \Rightarrow (F(X) \Rightarrow_1 K))$ and prove $\Psi \subset (\mathrm{H} \Rightarrow (\mu_1 X\, F(X) \Rightarrow_1 K))$. This comes from $\Delta_{\mathrm{H}}[\Delta_{\mathrm{H}}] \subset (\mu_1 X\, F(X) \Rightarrow_1 K)$, which is obtained using lemma 50 and $\forall X\, ((\Delta_{\mathrm{H}}[\Delta_{\mathrm{H}}] \subset (X \Rightarrow_1 K)) \Rightarrow \Delta_{\mathrm{H}}[\Delta_{\mathrm{H}}] \subset (F(X) \Rightarrow_1 K))$, which is obtained itself by a tedious but easy sequence of subtyping rules. ∎

Theorem 52 (fixpoint rule) *We derive the following rule where Y is a fixpoint combinator:*

$$\frac{\Gamma \vdash t : \forall X\, ((X \Rightarrow_1 K) \Rightarrow (F(X) \Rightarrow_1 K))}{\Gamma \vdash (Y\, t) : \mu_1 X\, F(X) \Rightarrow_1 K}$$

Proof: By corollary 37, we get $\vdash Y : \Psi$. By the previous lemma we have

$$\vdash Y : \forall X\, ((X \Rightarrow_1 K) \Rightarrow (F(X) \Rightarrow_1 K)) \Rightarrow (\mu_1 X\, F(X) \Rightarrow_1 K).$$

Then the result follows immediately. ∎

6.4 About data types.

In this section, we will see that we can prove in the system the usual results concerning data-types of system F or AF2 [21,22] and as a consequence prove subject reduction for these data-types. We will do the proof for the particular case of Church numerals, but it is clear that it can be extended to all the data types of system F or AF2. We will need to use a fixpoint operator. We defined above a fixpoint operator μ_1 for F of sort $(s \to \tau) \to (s \to \tau)$. It is clear we can also define a fixpoint μ_0 for F of sort $\tau \to \tau$, for instance by writing [2]:

$$\mu_0 X\, F(X) := \forall Y\, ((F(Y) \subset Y) \Rrightarrow Y)$$

Definition 53 *We define the following formulae of sort τ:*

$$\mathrm{N} := \forall X\, ((X \Rightarrow X) \Rightarrow X \Rightarrow X)$$
$$\mathrm{N}' := \Lambda F\, \Lambda X\, \mu_0 R\, (X \cup F[R])$$

[2] We could also get it from the previous fixpoint by writing $\mu_0 X\, F(X) := (\mu_1 R\, \lambda z F(R(z)))(\bot_o)$

N is the usual type of Church numerals. N′ is a way to define the set of all Church numerals using a fixpoint operator.

Fact 54 *We can prove both* $\vdash \mathrm{N} \subset \mathrm{N}'$ *and* $\vdash \mathrm{N}' \subset \mathrm{N}$

Proof: First we prove $\mathrm{N} \subset \mathrm{N}'$. Using inversion, it is enough to prove

$$\mathrm{N} \subset (F \Rightarrow X \Rightarrow \mu_0 R\,(X \cup F[R])).$$

Let us define $\Phi = \lambda R(X \cup F[R])$ and $\Theta = \mu_0 R\,\Phi(R)$. So we must prove $\mathrm{N} \subset (F \Rightarrow X \Rightarrow \Theta)$. This comes from $((\Theta \Rightarrow \Theta) \Rightarrow \Theta \Rightarrow \Theta) \subset (F \Rightarrow X \Rightarrow \Theta)$, which is a consequence of the three following subtyping judgments:

- $\Theta \subset \Theta$, which is immediate.
- $F \subset (\Theta \Rightarrow \Theta)$, which comes from $F[\Theta] \subset \Theta$ using fact 31, which comes from an easy but tedious sequence of subtyping rules.
- $X \subset \Theta$, which is easy using the fact that μ_0 is a fixpoint and Φ is increasing.

We now prove $\mathrm{N}' \subset \mathrm{N}$. It comes from $\mathrm{N}' \subset ((X \Rightarrow X) \Rightarrow X \Rightarrow X)$, which comes from $\mu_0 R\,(X \cup (X \Rightarrow X)[R]) \subset X$. We deduce it from the following trivial property of μ_0:

$$\forall F\,\forall K\,(\forall X_0\,((X_0 \subset K) \Rightarrow (F(X_0) \subset K)) \Rightarrow (\mu_0 X_0\ F(X_0) \subset K))$$

This means we assume $X_0 \subset X$ and prove $(X \cup (X \Rightarrow X)[X_0]) \subset X$. This is easy. ∎

Corollary 55 *Subject reduction for* β *holds for closed terms of type* N.

Proof: We assume $\vdash t : N$. By the previous fact we get $\vdash t : N'$. But it is easy to show that the interpretation of N' is the set of all λ-terms $\beta\eta$-equivalent to a λ-term of the form $\lambda f \lambda x\,(f^n\,x)$. Therefore, we have t $\beta\eta$-equivalent to $\overline{n} = \lambda f \lambda x\,(f^n\,x)$ for some n. Thus, any term t' obtained from t by β-reduction can be obtained from $\overline{n}$ by β-expansion which implies $\vdash t' : N$ because of theorem 41 and because $\vdash \overline{n} : N$. ∎

7 Further works.

7.1 Applications.

This paper lacks concrete examples of applications. Our idea is to design a small programming language where you can give typing information in your program (inspired for Hoare's logic [23]). Then a "compiler" should construct a proof from which we can extract this program together with a set of proof obligations for statement with no algorithmic content.

The main quality of the system, is its expressive power. For instance, if we want to program the primality test on integer, we can try to extract this program from the proof of two distinct statements:

- $\forall x(Nx \Rightarrow \exists b(Bb \restriction (b = 1 \Leftrightarrow \text{Prime}(x))))$
  This statement uses the standard form $\forall x(D(x) \Rightarrow \exists y(D'(y) \restriction S(x,y)))$ where D and D' are data-types and S is the specification of the program we want.
- $\forall x(Nx \Rightarrow (\text{Prime}(x)||\neg\text{Prime}(x)))$
  This second form is more concise and really uses the expressive power of the system.

However, in our paradigm, we start with a program! We do not know which kind of statement will be easier to manipulate to build a proof from which we can extract the program. So we should find a framework for our programming language where the full power of the system is accessible not to limit the possibility of the users.

This approach is identical to Parent's work [24] using the Calculus of Inductive Constructions, but we hope to have a more flexible (and thus powerful) way of annotating programs using the expressive power of our system.

7.2 Theoretical progress.

The problem of finding a finite axiomatization for β-reduction is studied and solved in [25] by adding a few more axioms and rules. Moreover, we also prove in this paper the completeness (using these new axioms) of system ST with respect to the realizability semantics we gave.

However, these axioms are not really useful to prove or extract a program. There are two reasons: the axioms given in this paper were the only needed for all the examples we tried, so in practice it really seems that adding new axioms is only necessary to get theoretical result on the system.

Another point is the fact that subject-reduction for β-reduction is not useful to prove a program! What you really need is subject-reduction for β-expansion!! Indeed, if you want to prove a property of a program t, you may need to give arguments to t and reduce it. But we do this moving "up" in the proof and this correspond to β-expansion! Subject-reduction for β-reduction would really correspond to a proof were we β-expand a program to prove it.

The main argument for the need for subject-reduction for β-reduction is the correctness of the system. But our semantics can play a similar role. Furthermore, we have seen in the section 6 that we can prove subject-reduction in the particular case of data-types which is really where it is needed: a program whose type says it computes a natural number will really compute a natural number.

Another question, is the proof theoretical strength of the propositional version of our system. Most of this paper only uses quantification on variables of sort τ. If we limit ourselves to that system, we can already define Church numeral and get the power of second order arithmetic (we can define zero and successor in the system, prove they are distinct and that successor is injective and we can write a definition of natural numbers implying the induction schema using the direct image definable in our system). We plan to make that idea more precise (at the moment the proof is only a sketch and may be wrong). But can we do better than second order arithmetic like higher-order logic?

7.3 Work on the axioms.

The extra axioms and rules can seem awkward as they were introduced one by one to get the wanted examples and later to get the completeness for realizability proved in [25].

This is not completely true: first the author tried hard to simplify the axioms as much as possible and even to prove them (if they remain, it is because we failed to derive them). In fact, all the axioms have a clear semantical meaning and therefore a simple soundness proof.

What is really needed is to study the independence of all the axioms and rules given in this paper and in [25]. This will probably leads to a simplification of the system. But this is not a simple task (except for one or two axioms or rules, for syntactic reasons) and here is an example showing why it is difficult:

If you want to study the set of axioms giving subject-reduction for β-reduction (either those in this paper giving a partial result or those in [25]), you need to study the difference between the two following semantics:

- The semantics where types are interpreted by sets of λ-terms closed for $\beta\eta$-equivalence (this is the semantics we used here).
- The semantics where types are interpreted by sets of λ-terms stable under β-expansion and η-equivalence only.

The problem is that it is very hard: as far as the author knows, nobody exhibited a closed type in system F (or system ST) whose interpretation is different in both semantics. Moreover, we know that both semantics coincide on $\forall$-positive types and that if they coincide on types A and B then they coincide on type $A \Rightarrow B$.

Proving that such a closed type exists would prove that some axioms or rules are really needed to get subject-reduction for β. This argument shows that this problem is hard and it is likely that if you introduce (as I do) more than one axiom for this, then it will be even harder to prove that they are independent one by one.

We are now working on this interesting problem ...

A Summary of the definitions and axioms.

A.1 Definitions.

$$\bot_\tau := \forall^\tau_\tau X\, X \qquad \text{(12 page 9)}$$

$$\bot_o := \forall^\tau_o X\, \forall^\tau_o Y\, (X \subset Y) \qquad \text{(12 page 9)}$$

$$\top_\tau := \forall^\tau_\tau K (\forall^\tau_o X (X \subset K) \Rrightarrow K) \qquad \text{(12 page 9)}$$

$$\top_o := \forall^\tau_o X\, (X \subset X) \qquad \text{(12 page 9)}$$

$$A \restriction P := \forall K\, ((A \subset (P \Rrightarrow K)) \Rrightarrow K) \qquad \text{(20 page 11)}$$

$$\bigcup x\, A(x) := \forall K\, (\forall x\, (A(x) \subset K) \Rrightarrow K) \qquad \text{(24 page 12)}$$

$$A[B] := \forall X\, ((A \subset (B \Rightarrow X)) \Rrightarrow X) \qquad \text{(29 page 13)}$$

$$A^{-1}[B] := \bigcup X\, (X \restriction (A \subset (X \Rightarrow B))) \qquad \text{(29 page 13)}$$

$$A \cap B := \bigcup X\, (X \restriction ((X \subset A) \wedge (X \subset B))) \qquad \text{(32 page 14)}$$

$$A \cup B := \forall K\, ((A \subset K) \Rrightarrow (B \subset K) \Rrightarrow K) \qquad \text{(32 page 14)}$$

$$P \mid\mid Q := \forall X\, ((P \Rrightarrow X) \Rightarrow (Q \Rrightarrow X) \Rightarrow X) \qquad \text{(32 page 14)}$$

$$\Lambda X\, A(X) := \forall X\, (X \Rightarrow A(X)) \qquad \text{(33 page 15)}$$

A.2 Axioms and extra rules.

$$\vdash \forall A, B\, (\forall x\, (A(x) \Rightarrow B(x)) \subset (\forall x\, A(x) \Rightarrow \forall x\, B(x))) \qquad \text{(6 page 6)}$$

$$\vdash \forall P\, \forall A, B\, ((P \Rrightarrow A \Rightarrow B) \subset (A \Rightarrow P \Rrightarrow B)) \qquad \text{(6 page 6)}$$

$$\vdash \forall A, B\, (A \subset (\bot_\tau \Rightarrow B)) \qquad \text{(14 page 9)}$$

$$\frac{\Gamma \vdash t : (P \Rrightarrow \bot_\tau) \Rightarrow \bot_\tau}{\Gamma \vdash P} \qquad \text{(14 page 9)}$$

$$\frac{\Gamma, x : A \vdash A \subset B}{\Gamma \vdash A \subset B} \qquad \text{(14 page 9)}$$

$$\forall F\, \forall A\, (\forall x\, (F(x) \Rightarrow A) \subset (\bigcup x\, F(x) \Rightarrow A)) \qquad \text{(26 page 12)}$$

References

[1] W. Howard, The formulae-as-types notion of construction, To H.B. Curry: Essays on combinatory logic, λ-calculus and formalism (1980) 479–490.

[2] T. Coquand, G. Huet, The calculus of construction, in: Information and Computation, 1988, pp. 241–262.

[3] J. Krivine, M. Parigot, Programming with Proofs, Inf. Process. Cybern. EIK 26 (3) (1990) 149–167.

[4] J. Krivine, Un algorithme non typable dans le système F, in: Comptes Rendus de l'Académie des Sciences de Paris, Vol. 304 Série I, 1987, pp. 123–126.

[5] S. Berardi, Pruning simply typed λ-terms, Journal of Logic and Computation 6 (1996) 663–681.

[6] P. Curmin, Marquage des preuves et extraction de programmes, Ph.D. thesis, Équipe de logique Université Paris VII (1999).

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

[8] P.-M. Christine, Extracting F_ω's programs from proofs in the Calculus of Constructions, in: Sixteenth Annual ACM Symposium on Principles of Programming Languages, ACM, Austin, 1989.

[9] P. Severi, N. Szasz, Studies of a Theory of Specifications with built-in Program Extraction, Journal of Automated Reasoning 27 (1).

[10] M. Parigot, Recursive programming with proofs, Theoretical Computer Science 94 (1992) 335–356.

[11] M. Hofmann, B. Pierce, Positive subtyping, Tech. rep., LFCS, report number 94-303 (1994).

[12] F. Potier, Type inference in the presence of subtyping: from theory to practice, Tech. rep., INRIA, report number:3483 (1998).

[13] D. Aspinall, Subtyping with Singleton Types, in: Computer Science Logic, CSL'94, Vol. 933 of Lecture Notes in Computer Science, Springer, 1995, pp. 1–15.

[14] M. Coppo, M. Dezani-Ciancaglini, A New Type Assignment System for Lambda-Terms, Archives for Mathematical Logic 19 (1978) 139–156.

[15] J. Courant, Strong Normalization with Singleton Types, in: S. V. Bakel (Ed.), Second Workshop on Intersection Types and Related Systems, Vol. 70 of LNCS, Elsevier, Copenhaguen, Danemark, 2002.

[16] R. Harper, C. Stone, Deciding Type Equivalence with Singleton Kinds, in: Symposium on Principles of Programming Languages, 2000, pp. 214–227.

[17] B. C. Pierce, Programming with Intersection Types, Union Types, and Polymorphism, Tech. Rep. CMU-CS-91-106, CMU (1991).
URL `citeseer.nj.nec.com/article/pierce91programming.html`

[18] C. Raffalli, The PhoX proof assistant version 0.8, software available on the Internet: http://www.lama.univ-savoie.fr/~raffalli/phox.html (2002).

[19] A. Church, A formulation of the simple theory of types, Journal of Symbolic Logic 5 (1940) 56–80.

[20] J. Mitchell, Polymorphic type inference and containment, Information and Computation 76 (1988) 211–249.

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

[22] M. Parigot, Programming with Proofs: a Second Order Type Theory, Lecture Notes in Computer Science 300, communication at ESOP.

[23] C. Hoare, An axiomatic Basis For Computer Programming, Communication of the ACM 12 (10) (1969) 576–580.

[24] C. Parent, Synthesizing proofs from programs in the Calculus of Inductive Constructions, in: Mathematics for Programs Constructions, Vol. 947 of Lecture Notes in Computer Science, Springer Verlag, 1995.

[25] C. Raffalli, System ST, β-reduction and completeness, prépublication du LAMA numéro 02-4d, available from the web page of the author (2001).