On the system F as a glue language for natural-language compositional-semantics

Christian Retoré

On the system **F** *as a glue language for natural-language compositional-semantics*

Du système **F** comme langage d'assemblage pour la sémantique compositionnelle du langage naturel

Christian RETORÉ (Université de Bordeaux, LaBRI & INRIA)

25th August 2011

Résumé Afin de modéliser dans un cadre compositionnel des phénomènes de pragmatique lexicale comme ceux étudiés par [6, 15] et plus particulièrement par Nicholas Asher dans [1], notre équipe a utilisé dans divers travaux [2, 12, 14, 13] le système **F** de Jean-Yves Girard (1971) [7, 9] pour construire les formules logiques représentant le sens des énoncés — là où d'autres auteurs comme [10] utilisent plutôt la théorie des types (1984) de Per Martin-Löf. [11] Nous expliquons dans cette note les raisons de notre préférence.

Abstract *In order to model in compositional framework some phenomena of lexical pragmatics as the ones studied by [6, 15] and especially by Nicholas Asher in [1], several contributions developed in our team [2, 12, 14, 13] did use the system* **F** *of Jean-Yves Girard (1971) [7, 9] to construct logical formulae expressing the meaning — while others like [10] prefer to use Per Martin-Löf's type theory [11]. In this note we explain the motivations supporting our preference.*

Qu'est-ce que le système F? Introduit dans [7], (une bonne référence est [9] — en anglais et disponible sur Internet — une autre est [8]), ce système est la logique propositionnelle intuitionniste du second ordre. On y quantifie sur les propositions: on prouve des formules du genre $\forall p\ (p \to p) \to (p \to p)$ noté $\Pi p.\ (p \to p) \to (p \to p)$ et les preuves s'écrivent comme des termes avec un Λp terme représentant la règle de quantification sur les types (introduction du $\Pi\ p$ ci-dessus), par exemple: $\Lambda p.\ \lambda f^{p\to p} \lambda x^p f(f(x))$ qui est du type $\Pi p.\ (p \to p) \to (p \to p)$ — la variable de type sur laquelle on quantifie ne doit pas être libre dans le type d'une variable libre. Symétriquement, un terme générique u de type $\Pi p.\ T[p]$ peut être spécialisé sur un type A: le terme correspondant s'écrit $u\{A\}$, et il est de type $T[A]$. Par exemple si u trie de liste de p-objets pour tout p, $u\{A\}$ trie des listes de A-objets. La beta réduction usuelle sur λ est $(\lambda x^T.\ u)t^T \rightsquigarrow u[x := t]$ mais celle sur Λ fait pareil: $(\Lambda\alpha.\ u)\{T\} \rightsquigarrow u[\alpha := T]$.

What is system* F*? *Introduced in [7], (a good reference freely available on the Web is [9] and another is [8]), this system is intuitionistic second order propositional logic. In this system one can quantify over propositions and can prove formula like* $\forall p\ (p{\to}p){\to}(p{\to}p)$ *written* $\Pi p.\ (p{\to}p){\to}(p{\to}p)$ *and proofs are written as terms with* Λp *denoting the rule introducing quantification over types — the type variable over which one quantifies should not be free in the type of a free variable. Symmetrically, a generic term* u *de type* $\Pi p.\ T[p]$ *can be specialised to a type* A*: the corresponding term, which is of type* $T[A]$ *is denoted by* $u\{A\}$*. For instance, if* u *sorts lists of* p*-objects for all* p*,* $u\{A\}$ *sorts lists of* A*-objects.*
Usual beta reduction concerns λ*:* $(\lambda x^T.\ u)t^T \rightsquigarrow u[x := t]$ *but beta reduction on* Λ *works just the same:* $(\Lambda\alpha.\ u)\{T\} \rightsquigarrow u[\alpha := T]$.

Capacité expressive On remarque facilement que la quantification universelle sur les types permet de définir toutes sortes d'opérateurs de constructions de types, notamment les plus courants, en particulier:

- le produit cartésien de deux types $A \times B \equiv \Pi X.(A \to (B \to X) \to X$
- la quantification existentielle sur les types $\exists p A[p] \equiv \Pi q.\ (\Pi p.\ (A[p] \to q)) \to q$
- les booléens $\Pi X.X \to X \to X$
- les entiers $\Pi X.(X \to X) \to (X \to X)$
- les listes d'objets de type α: $\Pi X.X \to (\alpha \to X \to X) \to X$
- ... d'où son succès pour ML, CaML etc.

On peut représenter toutes les fonctions récursives totales dont on peut prouver la totalité dans l'arithmétique de Peano du second ordre (qui pour ce genre d'énoncés coïncide avec l'arithmétique de Heyting).

Expressive power *Universal quantification over types let you define lots of type constructors, including the common ones, in particular:*

- *cartesian product* $A \times B \equiv \Pi X.(A \to (B \to X) \to X$
- *existential quantification over types* $\exists p A[p] \equiv \Pi q.\ (\Pi p.\ (A[p] \to q)) \to q$
- *booleans* $\Pi X.X \to X \to X$
- *integers* $\Pi X.(X \to X) \to (X \to X)$
- *lists of objects of type* α*:* $\Pi X.X \to (\alpha \to X \to X) \to X$
- *... hence its success with ML, CaML etc.*

A way to characterise is expressive power is to say that in this system one can define exactly all the total recursive functions whose totality can be proved in second order Peano arithmetic (classical or intuitionistic it makes no difference).

La logique d'assemblage n'est pas la logique où formuler la sémantique Le système **F** (logique propositionnelle du second ordre) sert de logique d'assemblage, mais il assemble les formules que l'ont veut, et ce peut bien sûr être des formules de la logique du premier ordre — même en utilisant des quantifications du second ordre, cela n'a pas de rapport. Un résultat dans la thèse de Bruno Mery [12] est le suivant: si les constantes du λ-calcul sont celles de la logique du premier ordre (resp. d'ordre n, d'ordre ω), alors les λ-termes de **F** normaux de type t sont les formules de la logique du premier ordre (resp. d'ordre n, d'ordre ω). Cela est bien sûr à rapprocher de la sémantique de Montague usuelle où le λ-calcul simplement typé (logique propositionnelle intuitionniste) avec deux types de base e et t permet d'exprimer et d'assembler les formules de la logique du premier ordre, d'ordre n ou d'ordre ω.

The glue logic has no relation with the logic used for formulating semantics *The system* **F** *(intuitionistic* second *order propositional logic) is used as a glue language, but it glues whichever formulae one likes. and it can be* first *order formulae — eventhough one uses second order quantifications, these are unrelated languages. A result in the PhD of Bruno Mery [12] garantees that when* λ*-calculus constants are the ones of first (resp.* n*,* ω *order logic, the normal* λ*-terms of* **F** *of type* t *are formulae of first (resp.* n*,* ω*) order logic. This is close to usual Montague semantics where simply typed* λ*-calculus (intuitionistic propositional logic) with two base types* e *and* t *is used to express and to glue formulae of first order logic,* n *order logic and even* ω *order logic.*

Sous-typage Comme j'en ai parfois fait la remarque, le sous-typage est plus ou moins incompatible le système **F** malgré diverses tentatives comme [4]. Mais je ne suis pas sûr que le sous-typage au sens de la programmation fonctionnelle soit ce dont on a besoin pour la sémantique dans le style de [15, 1]. Un calcul typé avec sous-typage permet, à partir d'inclusions sur les types atomiques, de dériver systématiquement les inclusions sur les types complexes. On peut se demander si, sémantiquement, les inclusions pertinentes de types composés ($a \to b$) sont toutes celles issues de celles sur a et de celles sur b. Je ne suis pas sur que le sous typage des verbes, par exemple, se déduise du sous-typage sur leurs arguments, sujet, objet, etc. La classification des aliments ou des "mangeurs" et des "aliments" induit-elle une classification des variantes de manger (*bouffer, dévorer, déguster*) ? Je ne crois pas. On veut donc plutôt veut un mécanisme de sous-typage indépendant sur chaque famille de types, et ce genre de "sous typage" n'a rien à voir avec une relation de sous-typage au sens de la programmation fonctionnelle. On peut même remarquer que la langue ne permet pas toutes les inclusions ontologiques, et qu'il n'est même pas clair que les inclusions arbitraires du langage définissent un ordre.

Subtyping *As I sometimes said, sub-typing is more or less incompatible with system* **F** *despite some attempts like [4] But I am not sure that subtyping in its functional-programming meaning is what we are looking for when dealing with the semantic questions of [15, 1]. A typed calculus with subtyping automatically derives inclusions between complex types from inclusions between base types. We should wonder whether it is semantically sound that inclusions between complex types like $a \to b$ are all the ones derived from inclusions on a and inclusions on b. For instance, I am not sure that subtyping on verb types derive from their arguments, subject, object, etc. Does classifications of "food"and "eaters" provide a classification of "eating" verbs (*swallow, taste, appreciate*)? I do not think so. We are rather looking for an independent "subtyping" mechanism for each type (or type family) and this kind of subtyping has nothing to do with the standard notion subtyping at work in functional programming. One can even observe that language does not allow all the ontological inclusions and that it is not so clear that the idiosyncratic linguistic inclusions define an order.*

Intérêt de la quantification sur les types C'est assez pratique. En particulier il n'y a qu'un quantificateur universel, qu'on spécialise, et non un par sorte. Sans le Λ il faudrait $\forall_a : (a \to t) \to t$ pour chaque type T, y compris pour les types complexes. Ici, un seul suffit: $\forall : \Pi a.\ ((a \to t) \to t)$, on l'applique ensuite à T pour obtenir le quantificateur sur T: $\forall\{T\} : ((T \to t) \to t)$. (On eput même avoir une constante $\forall : \Pi a.\ t$, sans *type raising*). De même on peut avoir un "il exists", un "la plus part" etc. qui ensuite sont spécialisés pour chaque type particulier. On peut aussi traiter dans ce cadre le ϵ ou τ de Hilbert qui sélectionne un individu satisfaisant une propriété dès qu'il en existe — notamment utilisé pour modéliser le voyageur fictif $\tau : \Pi a.\ ((a \to t) \to e)$. [14, 13]
On utilise des transformations et des instanciations sur des types composés, comme par exemple pour commuer un chemin en voyageur qui l'emprunte, en événement etc. Par exemple, comme ci-dessus, on peut opèrer sur le type raising, $(voie \to t) \to t$ qui devient $((humain \to t) \to t$. [14, 13]

Quantifying over types is great *At least, it is useful. In particular we have a single universal quantifier, that is specialised, and not one per sort. Without the Λ one would need $\forall_a : (a \to t) \to t$ for every type T, including the complex types. Here, one is enough: $\forall : \Pi a.\ ((a \to t) \to t)$, then it is applied to a type T to obtain the quantifier over T objects: $\forall\{T\} : ((T \to t) \to t)$. (One can even have a constant $\forall : \Pi a.\ t$, without type raising). Similarly one can have a "there exists", a "most of" that are specialised to every particular type. The very same trick can be used for Hilbert's ϵ and τ selecting an object satisfying a property as soon as there is one — we already used it, for instance, in the virtual traveller $\tau : \Pi a.\ ((a \to t) \to e)$. [14, 13]*
We use transformations and instantiations on complex types, for instance to turn a path into a traveller following it, into an event, etc. For instance, as above, we can operate on the type raising: $(path \to t) \to t$ qui devient $((human \to t) \to t$.

Questions de complexité En syntaxe avoir une classe de grammaires la plus petite possible permet d'avoir des algorithmes d'analyse plus performants. En sémantique, il n'en est rien: que doit faire au minimum la logique d'assemblage? on n'en sait rien... mais si on le savait cela améliorerait-il la complexité algorithmique de l'analyse sémantique? Non, pas du tout! Ce sont plutôt les choix des différents "sens" de chaque mot qui rendent le processus complexe. Le processus d'analyse est l'analyse syntaxique que vous préférez (et là il y a des questions de complexité), puis on construit un λ-terme avec les λ-termes sémantiques du lexique (un choix dont la complexité ne saurait être améliorée) et les règles syntaxiques, et on le réduit, et bien évidemment on n'a pas de beta réduction problématique — par exemple nécessitant un nombre exponentiel d'étapes de beta réduction en fonction de la longueur du terme à beta réduire.

Complexity issues *In syntax, having a class of grammars as restricted as possible, allows more efficient parsing algorithms. In semantics, it does not make sense: what is the minimum that the glue logic should do? No one knows, but even if this was known, would it help to reduce the algorithmic complexity of semantic analysis? Not at all! The complexity is rather due to the choice, for each word, of its relevant meaning among the possibilities provided by the lexicon. The process of analysis is simply syntactic analysis, and the syntactic analysis of a phrase can be the one you prefer (here is a complexity issue), then a λ-term is constructed following the syntax with the chosen semantic λ-terms from the lexicon (a choice whose complexity cannot be improved), then it is reduced and obviously the reduction is unproblematic — one does not get a term whose number of reduction steps is exponential with respect to the length of the term.*

Comparatif entre la théorie des types de Martin-Löf et le système F pour la sémantique Certains, comme Zhaohui Luo [10], utilisent plus volontiers la théorie des types de Martin-Löf (TT) dont il existe plusieurs variantes, l'une des premières étant [11]. Une discussion avec un expert des deux systèmes, Thierry Coquand (qui a étendu $\mathbf{F}_\omega$ aux types dépendants, pour faire le calcul des constructions [5] et qui depuis près de vingt ans travaille sur TT) a achevé de me convaincre: comme logique d'assemblage il vaut mieux choisir le système **F** pour sa simplicité formelle (4 règles) et conceptuelle (une seule sorte de type, un seul genre de jugements), car on n'a aucun problème de complexité dans la réduction — comme dit plus haut, on ne va pas chercher les fonctions à croissance ultra rapide du système **F** pour assembler des formules.
Si on prend une version faible de TT, celle-ci est, du point de vue des fondements — reconnaissons-le — plus simple. La normalisation des termes du système, permet d'établir sans trop de peine la cohérence du système (s'il existait une preuve du faux il en existerait une qui soit normale, et on peut voir assez facilement qu'il n'y en a pas). Pour TT il faut le schéma de compréhension (qui permet de définir un ensemble à partir d'une formule $\{X|P(X)\}$ pour les formule Pi^1_1 c'est-à-dire pour les formules de la forme "Pour tout ensemble X Il existe un ensemble Y..." tandis que pour **F** le schéma de compréhension pour toutes les formules.

Comparing Martin-Löf's type theory and system* F *for semantics *Some, like Zhaohui Luo [10] rather us Martin-Löf's type theory (TT), a system with several variants, and early one being in [11]. A converging discussion with an expert in both, namely Thierry Coquand (who extended $\mathbf{F}_\omega$ to dependent types, yielding the calculus of constructions [5], working with TT for twenty years) achieved to convince me: as a glue logic, system* **F** *should be preferred, because of its formal complexity, because of its conceptual simplicity (a single sort of types and a single sort of de judgements), because we have no complexity in reduction — as earlier said, we are not going to use extremely fast growing functions and the like when glueing formulae.*
If one uses a weak variant of TT, this system is admittedly simpler as far as foundations are concerned. Indeed, coherence is derived from normalisation (if there were a proof of a false statement, there would be a normal proof, but it is easily seen that there cannot be a normal proof of something false). For a weak variant of TT, the comprehension axiom (which defines a set $\{X|P(X)\}$ from a formula P) is needed for every Π^1_1 formula P (formulae like "for every set X there exists a set Y....."). For system **F** *the comprehension axiom is needed for every formula.*

***Records* et types dépendants** Quant au type de données populaire *records*, il s'agit simplement d'un produit (définissable dans dont on nomme les projections, un peu comme les attributs d'une base de données. Ils peuvent aussi se voir comme un cas particulier d'une famille d'ensembles indexée par un autre ensemble. Ce sont alors donc des types dépendants particuliers, et certains pensent que, d'une manière générale, les types dépendants sont utiles à la sémantique compositionnelle, ce qui, personnellement, ne me saute pas aux yeux. Néanmoins, pour les inconditionnels des types dépendants, ou si quelqu'un en montrait l'utilité, on peut les ajouter au système **F**. On peut même les ajouter à F_ω ce qui donne un système bien connu: le calcul des constructions de Coquand-Huet, implémenté dans l'assistant de preuve Coq. [5, 3]

Records and dependent types *Regarding the popular data type known as "records", they are just products with named projections, a bit like attributes in a data base. They can be viewed as particular indexed family of sets. Hence they are particular dependent types, and some thinks that dependent types are relevant for compositional semantics, also personnally I am not convinced. In any case, if one is keen on dependent types, or if one show they are useful at semantics, they could be added to system* **F**. *They can even be added to* F_ω *yielding a famous system: the calculus of constructions of Coquand-Huet, implemented in the Coq proof assistant. [5, 3]*

References

[1] Nicholas Asher. *Lexical Meaning in context – a web of words.* Cambridge University Press, 2011.

[2] Christian Bassac, Bruno Mery, and Christian Retoré. Towards a Type-Theoretical Account of Lexical Semantics. *Journal of Logic Language and Information*, 19(2):229–245, April 2010.

[3] Yves Bertot and Pierre Castéran. *Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions.* Texts in Theoretical Computer Science. Springer Verlag, 2004.

[4] Luca Cardelli, Simone Martini, John C. Mitchell, and Andre Scedrov. An extension of system F with subtyping. *Information and Computation*, 109(1/2):4–56, 1994.

[5] Thierry Coquand and Gérard P. Huet. The calculus of constructions. *Information and Computation*, 76(2/3):95–120, 1988.

[6] D.A. Cruse. *Lexical semantics.* Cambridge textbooks in linguistics. Cambridge University Press, 1986.

[7] Jean-Yves Girard. Une extension de l'interprétation de Gödel à l'analyse et son application: l'élimination des coupures dans l'analyse et la théorie des types. In Jens Erik Fenstad, editor, *Proceedings of the Second Scandinavian Logic Symposium*, volume 63 of *Studies in Logic and the Foundations of Mathematics*, pages 63–92, Amsterdam, 1971. North Holland.

[8] Jean-Yves Girard. *Le point aveugle. Cours de logique. 1: Vers la perfection. 1.* Paris: Hermann, Éditeurs des Sciences et des Arts. xvi, 280 p. , 2006.

[9] Jean-Yves Girard, Yves Lafont, and Paul Taylor. *Proofs and Types.* Number 7 in Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1988.

[10] Zhaohui Luo. Contextual analysis of word meanings in type-theoretical semantics. In Sylvain Pogodalla and Jean-Philippe Prost, editors, *LACL*, volume 6736 of *Lecture Notes in Computer Science*, pages 159–174. Springer, 2011.

[11] Per Martin-Löf. *Intuitionistic Type Theory.* Bibliopolis, Napoli, 1984. (Lecture Notes by G. Sambin).

[12] Bruno Mery. *Modélisation de la sémantique lexicale dans le cadre de la théorie des types.* PhD thesis, Université de Bordeaux 1, July 2011.

[13] Richard Moot, Laurent Prévot, and Christian Retoré. A discursive analysis of itineraries in an historical and regional corpus of travels. In *Constraints in discourse*, page http://passage.inria.fr/cid2011/doku.php, Ayayroches-rouges, France, September 2011. Projet ITIPY de la région Aquitaine.

[14] Richard Moot, Laurent Prévot, and Christian Retoré. Un calcul de termes typés pour la pragmatique lexicale. In *Traitement Automatique du Langage Naturel, TALN 2011*, pages 161–166, Montpellier, France, June 2011. Projet ITIPY de la région Aquitaine.

[15] James Pustejovsky. *The generative lexicon.* M.I.T. Press, 1995.