

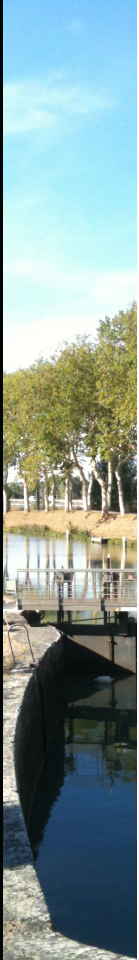
# Sémantique du langage naturel en théorie des types

**Christian Retoré**

Université de Bordeaux & IRIT (Toulouse)

Séminaire TEXTE

LIRMM, Montpellier, 14 nov. 2012



## **A Foreword – contents**

## A.1. Merci... et... sorry!

... au CNRS et à l'IRIT (Toulouse),  
qui me donnent d'excellentes conditions de recherche  
(mais pour un an seulement)

... à Violaine Prince et à Jean-Philippe Prost de m'avoir invité.

**Désolé, les transparents sont en anglais  
(des passages sont repris de mon cours à ESSLLI)**





## A.2. Contents, roughly speaking

All time favourites:

- simply typed lambda calculus

- formulae in simply typed lambda calculus

- Montague semantics

Extending the type system to second order

- second order typed lambda calculus

- modelling lexical pragmatics

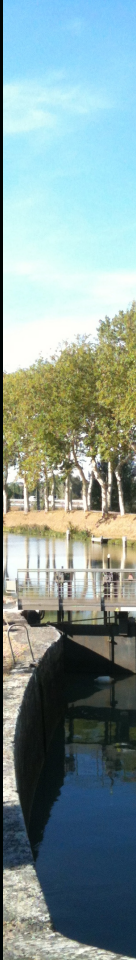
- a language for quantification, plurals,

Questions:

- base types? acquisition

- multi sorted logic vs. type theory

- sub-typing (specialisation relations)



**B Logical formulae as simply  
typed lambda terms and  
compositional semantics**



## B.1. Mind that there are TWO logics

One logic for expressing **meanings:**  
**formulae** of first or higher order logic, single or multi sorted.

One logic for **meaning assembly:**  
**proofs** in propositional logic,  
of the well-formedness of formulae.

## B.2. Natural deduction / simply typed lambda calculus

Proofs are trees of formulae,

- leaves are hypothesis (that may be cancelled or not)
- the root is the conclusion
- branching are rules

plus the indication of the rule which cancelled a cancelled hypothesis.



### B.3. Types and terms: Curry-Howard

	Introduction rules	Elimination rules
Implication	$\frac{\begin{array}{c} \Gamma[A]_{\alpha} \Delta[A]_{\alpha} \\ \vdots \\ B \end{array}}{A \rightarrow B} \rightarrow_i$ <p><i>some As are cancelled.</i></p>	$\frac{A \quad A \rightarrow B}{B} \rightarrow_e$

We shall focus on the  $\rightarrow$  connective.

A proof of  $A \rightarrow B$  is a function that maps proofs of  $A$  to proofs of  $B$ .

Think of a formula/type as the set of its proofs.



## B.4. Terms

Types are.... formulae.

$\lambda$ -terms encode proofs  $u : U$  means  $u$  is a term of type  $U$ .

Inside term one rather writes  $u^U$ .

1. hypotheses we have a countable set of variables of each type which are terms of this type
2. constants = never cancelled hypotheses there can be as many constants of each type as needed
3.  $\rightarrow$  introduction if  $x : U$  is a **variable** and  $t : T$  then  $(\lambda x^U. t) : U \rightarrow V$ .
4.  $\rightarrow$  elimination if  $f : U \rightarrow V$  and  $t : U$  then  $f(t) : V$

With such typed terms we can faithfully encode proofs.

Variables are hypotheses (that are simultaneously cancelled).

## B.5. Reduction

A redex is the sequence of an introduction rule of the connective  $C$  followed by an elimination of the same connective  $C$ .

In the  $\rightarrow$  fragment there is a single reduction rules for  $\rightarrow$ :

$$(\lambda x : U. t)^{U \rightarrow V} u^U \text{ reduces to } t[x := u] : V$$

Every proof /  $\lambda$ -term reduces to a normal form (weak normalisation) no matter how one proceeds (strong normalisation).

Normalisation is confluent or Church-Rosser (not difficult but tedious).

So every lambda term reduces to a unique normal lambda term.

## B.6. Categorical interpretation of simply typed lambda calculus

A way to implement to prove coherence:

define a category (a CCC) with

formulae/ types: structured sets defined from arbitrary structured sets corresponding to propositional variables / base types

morphisms: functions preserving the structure

in such a way that a closed term  $t$  of type  $A$  to  $B$  corresponds to a function  $[t]$  from  $A$  to  $B$

whenever  $t$  reduces to  $t'$  the functions  $[t]$  and  $[t']$  are the same

## B.7. Representing multi sorted formulae within lambda calculus — connectives

Assume that the base types are

$e_i$  (sorts for individuals, often there is just one) and  
 $t$  (propositions)

and that the only constants are

the logical ones (below) and

the relational and functional symbols of the specific logical language (on the next slide).

We need the logical constants:

- $\sim$  of type  $t \rightarrow t$  (negation)
- $\supset, \&, +$  of type  $t \rightarrow (t \rightarrow t)$  (implication, conjunction, disjunction)
- for each  $i$  two constants  $\forall$  and  $\exists$  of type  $(e_i \rightarrow t) \rightarrow t$

## B.8. Representing formulae within lambda calculus — language constants

The language constants for multi sorted First Order Logic:

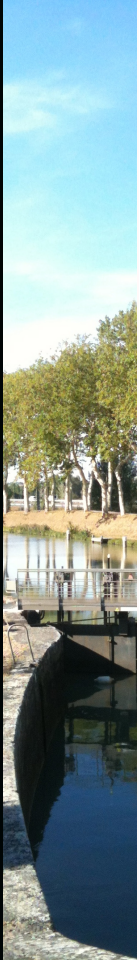
- $R_q$  of type  $\mathbf{e}_{q_1} \rightarrow (\mathbf{e}_{q_2} \rightarrow (\dots \rightarrow \mathbf{e}_{q_{sq}} \rightarrow \mathbf{t}))$   
e.g. likes:  $e \rightarrow e \rightarrow t$  sleeps  $e \rightarrow t$
- $f_q$  of type  $\mathbf{e}_{q_1} \rightarrow (\mathbf{e}_{q_2} \rightarrow (\dots \rightarrow \mathbf{e}_{q_{sq}} \rightarrow \mathbf{e}_{q_0}))$

## B.9. From formulae to normal lambda terms

A rather simple exercise: follow the structure of the formulae.

Examples:  $\forall x. \textit{barber}(x) \supset \textit{shaves}(x, x)$

$$\forall(\lambda x^e. (\supset \textit{barber}(x))((\textit{shaves}(x))(x)))$$



## B.10. Extension: higher order logic...

A three place predicate variable:  $X : (\mathbf{e} \rightarrow \mathbf{e} \rightarrow \mathbf{e} \rightarrow \mathbf{t})$

The quantifier over such variables:  $\forall_1^3 : (\mathbf{e} \rightarrow \mathbf{e} \rightarrow \mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$

*w has any property that an idiot has.*

$$\forall P(\exists x \text{ stupid}(x) \text{ and } P(x)) \supset P(w)$$

Exercise: write it in lambda calculus.

## B.11. From normal lambda terms to formulae

This relies on the (long) normal form of lambda terms.

We assume a refinement, that they are beta eta long: long  $\beta\eta$ -normal forms (Huet76) which are defined using the auxiliary notion of atomic forms:

- Variables are atomic form.
- $(M N)$  is an atomic form when  $M : A \rightarrow B$  is an atomic term and  $N : A$  is in long  $\beta\eta$ -normal form.
- Atomic forms whose type is a base type are long  $\beta\eta$ -normal forms.
- $\lambda x^A. t$  is a long  $\beta\eta$ -normal form whenever  $t$  is one as well.



## B.12. Why using long $\beta\eta$ -normal forms?

Every term as a unique long  $\beta\eta$ -normal form, which is obtained by beta reduction and then some eta expansion steps.

This makes sure that inside a normal lambda term whenever a subterm is of type  $A \rightarrow B$  its is a lambda abstraction. (Sub)terms like  $\forall sleep^{e \rightarrow t}$  are avoided.

This is achieved by replacing

any sub term like  $likes^{e \rightarrow e \rightarrow t}$  without argument with  $\lambda x^e. \lambda y^e. ((likesx)y)$

and a sub term  $(\forall sleep) : t$  with  $\forall (\lambda x^e (sleep(x)))$ .



## B.13. The shape of normal lambda terms

$$\lambda x_{i_1} \cdots \lambda x_{i_n} \cdot h t_1 \cdots t_p$$

Possibly  $n = 0$  no  $\lambda$  ahead.

$h$ : is a variable or a constant of type  $U_1 \rightarrow \cdots \rightarrow U_p \rightarrow X$

The  $t_i$  are themselves **normal**  $\lambda$ -terms

The variable/constant  $h$  correspond to the hypothesis containing the conclusion as a subformula —as observed from the subformula property.

## B.14. Normal terms of type $t$ are formulae

This can be shown by structural induction on the shape of the  $\lambda$ -term, e.g. they cannot be any head  $\lambda$ s, and proceed by cases on the head variable/constant:

1. normal  $\lambda$ -terms of type  $e_i$  with  $x_k : e_j$  as only free variables are logical terms with the same free variables
2. normal  $\lambda$ -terms of type  $t$  with  $x_i : e$  as only free variables are logical formulae with the same free variables and bound variables.

The long  $\eta\beta$  normal form replaces say  $f^{a \rightarrow b}$  by  $\lambda x:a(fx)$  to have a better correspondence between the type and formula variables since every arrow-typed term starts with a  $\lambda$ .

## B.15. Remarks in view of Montague semantics

Non normal lambda terms of type  $t$  coming from syntax do not really correspond to formulae.

Hence we need:

- normalisation
- a proof that the normal terms do correspond to formulae, as we just shown.



## B.16. Back to the roots: Montague semantics. Types.

Simply typed lambda terms

$$\text{types} ::= e \mid t \mid \text{types} \rightarrow \text{types}$$

*chair* , *sleep*  $e \rightarrow t$

*likes* transitive verb  $e \rightarrow (e \rightarrow t)$

## B.17. Back to the roots: Montague semantics. Syntax/semantics.

(Syntactic type)*	=	Semantic type
$S^*$	=	$t$ a sentence is a proposition
$np^*$	=	$e$ a noun phrase is an entity
$n^*$	=	$e \rightarrow t$ a noun is a subset of the set of entities
$(A \setminus B)^* = (B / A)^*$	=	$A \rightarrow B$ extends easily to all syntactic categories of a Categorical Grammar e.g. a Lambek CG

## B.18. Back to the roots: Montague semantics. Logic within lambda-calculus 1/2.

Logical operations (and, or, some, all the,.....) need constants:

Constant	Type
$\exists$	$(e \rightarrow t) \rightarrow t$
$\forall$	$(e \rightarrow t) \rightarrow t$
$\wedge$	$t \rightarrow (t \rightarrow t)$
$\vee$	$t \rightarrow (t \rightarrow t)$
$\supset$	$t \rightarrow (t \rightarrow t)$

## B.19. Back to the roots: Montague semantics. Logic within lambda-calculus 2/2.

Words in the lexicon need constants for their denotation:

<i>likes</i>	$\lambda x \lambda y (\text{likes } y) x$	$x : e, y : e, \text{likes} : e \rightarrow (e \rightarrow t)$
<< likes >> is a two-place predicate		
<i>Garance</i>	$\lambda P (P \text{ Garance})$	$P : e \rightarrow t, \text{Garance} : e$
<< Garance >> is viewed as the properties that << Garance >> holds		





## B.20. Back to the roots: Montague semantics. Computing the semantics. 1/5

1. Replace in the lambda-term issued from the syntax the words by the corresponding term of the lexicon.
2. Reduce the resulting  $\lambda$ -term of type  $t$  its normal form corresponds to a formula, the "meaning".

## B.21. Back to the roots: Montague semantics. Computing the semantics. 2/5

<b>word</b>	<b><i>semantic type</i> <math>u^*</math></b> <b><i>semantics</i> : <math>\lambda</math>-term of type <math>u^*</math></b> $x_v$ <i>the variable or constant <math>x</math> is of type <math>v</math></i>
some	$(e \rightarrow t) \rightarrow ((e \rightarrow t) \rightarrow t)$ $\lambda P_{e \rightarrow t} \lambda Q_{e \rightarrow t} (\exists_{(e \rightarrow t) \rightarrow t} (\lambda x_e (\wedge_{t \rightarrow (t \rightarrow t)} (P x)(Q x))))$
statements	$e \rightarrow t$ $\lambda x_e (\text{statement}_{e \rightarrow t} x)$
speak about	$e \rightarrow (e \rightarrow t)$ $\lambda y_e \lambda x_e ((\text{speak\_about}_{e \rightarrow (e \rightarrow t)} x)y)$
themselves	$(e \rightarrow (e \rightarrow t)) \rightarrow (e \rightarrow t)$ $\lambda P_{e \rightarrow (e \rightarrow t)} \lambda x_e ((P x)x)$

## B.22. Back to the roots: Montague semantics. Computing the semantics. 3/5

The syntax (e.g. a Lambek categorial grammar) yields a  $\lambda$ -term representing this deduction simply is

*((some statements) (themselves speak\_about))* of type  $t$

## B.23. Back to the roots: Montague semantics. Computing the semantics. 4/5

$$\begin{aligned}
 & \left( (\lambda P_{e \rightarrow t} \lambda Q_{e \rightarrow t} (\exists_{(e \rightarrow t) \rightarrow t} (\lambda x_e (\wedge (P x) (Q x)))) \right) \\
 & \quad \left( \lambda x_e (\text{statement}_{e \rightarrow t} x) \right) \\
 & \quad \left( (\lambda P_{e \rightarrow (e \rightarrow t)} \lambda x_e ((P x)x) \right) \\
 & \quad \left( \lambda y_e \lambda x_e ((\text{speak\_about}_{e \rightarrow (e \rightarrow t)} x)y) \right)
 \end{aligned}$$

$\downarrow \beta$

$$\begin{aligned}
 & (\lambda Q_{e \rightarrow t} (\exists_{(e \rightarrow t) \rightarrow t} (\lambda x_e (\wedge_{t \rightarrow (t \rightarrow t)} (\text{statement}_{e \rightarrow t} x) (Q x)))) \\
 & \quad \left( \lambda x_e ((\text{speak\_about}_{e \rightarrow (e \rightarrow t)} x)x) \right)
 \end{aligned}$$

$\downarrow \beta$

$$(\exists_{(e \rightarrow t) \rightarrow t} (\lambda x_e (\wedge (\text{statement}_{e \rightarrow t} x) ((\text{speak\_about}_{e \rightarrow (e \rightarrow t)} x)x))))$$

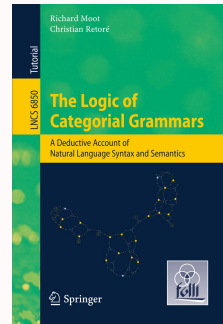
## B.24. Back to the roots: Montague semantics. Computing the semantics. 5/5

This term represent the following formula of predicate calculus  
(in a more pleasant format):

$$\exists x : e (\text{statement}(x) \wedge \text{speaks\_about}(x, x))$$

This is a (simplistic) semantic representation of the analysed  
sentence.

Advertisement: for more details on classical  
Montague semantics see chapter 3 of "The  
logic of categorial grammars" (Moot, Retoré  
Springer, 2012).



## B.25. Good overall architecture / limits

Good trick (Church): a propositional logic for meaning assembly (proofs/ $\lambda$ -terms) to compute formulae another logic with first order (formulae/meaning no proofs)

Of course, we would like to use many sorts to reject:

\* The chair barks.

"barks" requires a subject of type "dog".

Many types (how many) ?

If we do not want too many operations, we need to factor similar operations acting on types.





## **C Second order lambda calculus (Girard's system F)**

## C.1. Types

- Constants types  $e_i$  and  $t$ , as well as any type variable  $\alpha, \beta, \dots$  in  $P$ , are types.
- Whenever  $T$  is a type and  $\alpha$  a type variable which may but need not occur in  $T$ ,  $\Lambda\alpha. T$  is a type.
- Whenever  $T_1$  and  $T_2$  are types,  $T_1 \rightarrow T_2$  is also a type.



## C.2. Terms

- A variable of type  $T$  i.e.  $x : T$  or  $x^T$  is a *term*.  
Countably many variables of each type.
- $(f \tau)$  is a term of type  $U$  whenever  $\tau : T$  and  $f : T \rightarrow U$ .
- $\lambda x^T. \tau$  is a term of type  $T \rightarrow U$  whenever  $x : T$ , and  $\tau : U$ .
- $\tau\{U\}$  is a term of type  $T[U/\alpha]$  whenever  $\tau : \Lambda\alpha. T$ , and  $U$  is a type.
- $\Lambda\alpha. \tau$  is a term of type  $\Lambda\alpha. T$  whenever  $\alpha$  is a type variable, and  $\tau : T$  without any free occurrence of the type variable  $\alpha$ . (Type of  $x$  in  $\forall\alpha. x^\alpha$ ???)

### C.3. Extension Curry-Howard

Types are quantified propositional formulae.

Terms are proofs in intuitionistic quantified propositional calculus.

Restriction  $q \rightarrow \alpha$  and  $q$  certainly yield  $\alpha$ , but fortunately from this one cannot conclude that under assumption  $q$  one has  $\forall \alpha. \alpha$ .

## C.4. Reduction

The reduction is defined as follows:

- $(\Lambda\alpha.\tau)\{U\}$  reduces to  $\tau[U/\alpha]$  (remember that  $\alpha$  and  $U$  are types).
- $(\lambda x.\tau)u$  reduces to  $\tau[u/x]$  (usual reduction).

Reduction is strongly normalising and confluent (Girard, 1971): *every term of every type admits a unique normal form which is reached no matter how one proceeds.*

(Difficult: the proof quantifies over all subsets of terms that behaves like strongly normalising terms)

## C.5. Unnecessary type operators

The following defined types have the same elimination and introduction behaviour.

Product  $A \wedge B$  can be defined as

$$\prod \alpha. (A \rightarrow B \rightarrow \alpha) \rightarrow \alpha$$

Sum  $A \vee B$  can be defined as

$$\prod \alpha. (A \rightarrow \alpha) \rightarrow (B \rightarrow \alpha) \rightarrow \alpha$$

Existential quantification:

$$\sum \beta. ((\prod \alpha. (V[X] \rightarrow \beta)) \rightarrow \beta)$$

## C.6. Inductive types (cf. ML, CaML, Haskell)

Integers

$$\Pi\alpha. \alpha \rightarrow (\alpha \rightarrow \alpha) \rightarrow \alpha$$

List of  $\beta$  objects. (the  $\beta$  can be quantified as well)

$$\Pi\alpha. \alpha \rightarrow (\beta \rightarrow \alpha \rightarrow \alpha) \rightarrow \alpha$$

Binary trees

$$\Pi\alpha. \alpha \rightarrow (\alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow \alpha$$

Binary trees with leaves  $L$  and nodes  $N$

$$\Pi\alpha. (L \rightarrow \alpha) \rightarrow (N \rightarrow \alpha \rightarrow \alpha \rightarrow \alpha) \rightarrow \alpha$$

## C.7. What can be defined, computed

The functions that can be programmed are the ones that can be proved total in second order Heyting arithmetic, that for such issues has the same power as second order Peano arithmetic.

All data types can be defined, and for such types their only normal terms are the expected ones.

More than polymorphic typed functional languages but every program terminates — there is no fixed point operator  $Y : \Pi\alpha. (\alpha \rightarrow \alpha) \rightarrow \alpha$ .



## C.8. Coherence, categorical interpretation and normalisation

System F may seem unsafe: one can define a type via reference to all types, including itself. Indeed, some proposed extension collapse.

An argument is strong normalisation since there is no normal proof of  $\perp = \prod \alpha. \alpha$  (relies on the comprehension axiom for all formulae of  $\text{HA}_2$ ).

Another argument is a concrete categorical interpretation (e.g. with coherence spaces) that shows that there are distinct functions from  $A$  to  $B$ .

**Every term reduces anyhow to its unique normal form.  
Terms of type  $t$  with constants of multisorted FOL (resp. HOL) correspond to multisorted formulae of FOL (resp. HOL)**



## **D System F based semantics and pragmatics**



## D.1. Examples

Dinner was delicious but took ages. (event / food)

\* The salmon we had for lunch was lightning fast. (animal / food)

I forgot on the table my preferred book on logic. (physical / info)

I carried the books from the shelf to the attic since i already read them. (phys. / info)

Liverpool is a poor town and an important harbour. (people / geographic)

\* Liverpool defeated Chelsea and is an important harbour. (football / geographic)

(Nevertheless: Barcelona won four champions leagues and organised the olympiads. Libourne, a small south-west town, defeated Lille.)



## D.2. Types and terms: system F

System F with many base types  $e_i$  (many sorts of entities)

$v$  (for events who play a particular role in  $\lambda$ -DRT)

$t$  truth values

types variables roman upper case, greek lower case

usual terms that we saw, with constants (free variables that cannot be abstracted)

Every normal terms of type  $t$  with free variables being logical variables (of a the corresponding multi sorted logic  $L$ ) correspond to a formula of  $L$ .

### D.3. Examples of second order usefulness

Arbitrary modifiers:  $\Lambda\alpha\lambda x^A y^\alpha f^{\alpha\rightarrow R}.((\text{read}^{A\rightarrow R\rightarrow t} x) (f y))$

Given types  $\alpha$ ,  $\beta$  and  $\gamma$

three predicates  $P^{\alpha\rightarrow t}$ ,  $Q^{\beta\rightarrow t}$ ,  $R^{\gamma\rightarrow t}$ ,

over entities of respective kinds  $\alpha$ ,  $\beta$  and  $\gamma$

for any  $\xi$  with three morphisms from  $\xi$  to  $\alpha$ , to  $\beta$ , and to  $\gamma$

we can coordinate the properties  $P, Q, R$  of (the three images of) an entity of type  $\xi$ :

$$\begin{aligned} \mathbf{AND3} = & \Lambda\alpha\Lambda\beta\Lambda\gamma \\ & \lambda P^{\alpha\rightarrow t}\lambda Q^{\beta\rightarrow t}\lambda R^{\gamma\rightarrow t} \\ & \Lambda\xi\lambda x^\xi \\ & \lambda f^{\xi\rightarrow\alpha}\lambda g^{\xi\rightarrow\beta}\lambda h^{\xi\rightarrow\gamma}. \\ & (\text{and}(\text{and} (P (f x))(Q (g x)))(R (h x))) \end{aligned}$$

## D.4. Definite determiner e.g. "the": à la von Heusinger

We use a selection operator  $\iota : (\Pi\alpha. (\alpha \rightarrow \mathbf{t}) \rightarrow \alpha$  which picks up (according to pragmatics considerations) an object in a class defined by a property say  $P$  of objects of type  $\tau$ .

Thus  $\iota\{\tau\}P$  is an object of type  $\tau$  —fine! but we need to say that this object enjoys  $P$ : that is:  
 $P(\iota\{\tau\}P)$ , which is added as a presupposition a universal one.

Remark 1: in  $\lambda$ -DRT this proposition can be propagated to the top level via the operator  $\oplus$ ).

Remark 2: "the" acts on a type, it is also possible to view "the" as a constant *iota* of type  $\forall\alpha.\alpha$ . (observe that this makes any type derivable and habited, but one can say "the unicorn").

## D.5. Principles of our lexicon

- Remain within realm of Montagovian compositional semantics (for compositionality)
- Allow both predicate and argument to contribute lexical information to the compound.
- Integrate within existing discourse models ( $\lambda$ -DRT).

We advocate a system based on *optional modifiers*.



## D.6. The Terms: main / standard term

- A standard  $\lambda$ -term attached to the main sense:
  - Used for compositional purposes
  - Comprising detailed typing information
  - Including slots for optional modifiers
  - e.g.  $\lambda\alpha\lambda\beta\lambda x^\alpha y^\beta f^{\alpha\rightarrow A} g^{\beta\rightarrow F} . ((\text{eat}^{A\rightarrow F\rightarrow t} (f\ x)) (g\ y))$
  - e.g. Paris<sup>T</sup>

## D.7. The Terms: Optional Morphisms

- Each a one-place predicate
- Used, or not, for adaptation purposes
- Each associated with a constraint : *rigid*,  $\emptyset$

$$* \left( \frac{Id^{F \rightarrow F}}{\emptyset}, \frac{f_{grind}^{Living \rightarrow F}}{rigid} \right)$$

$$* \left( \frac{Id^{T \rightarrow T}}{\emptyset}, \frac{f_L^{T \rightarrow L}}{\emptyset}, \frac{f_P^{T \rightarrow P}}{\emptyset}, \frac{f_G^{T \rightarrow G}}{rigid} \right)$$

## D.8. A Complete Lexical Entry

Every lexeme is associated to an  $n$ -uple such as:

$$\left( \text{Paris}^T, \frac{\lambda_x^T \cdot x^T}{\emptyset}, \frac{\lambda_x^T \cdot (f_L^{T \rightarrow L} x)}{\emptyset}, \frac{\lambda_x^T \cdot (f_P^{T \rightarrow P} x)}{\emptyset}, \frac{\lambda_x^T \cdot (f_G^{T \rightarrow G} x)}{\text{rigid}} \right)$$



## D.9. RIGID vs flexible use of optional morphisms

Type clash:  $(\lambda x^V. (P^{V \rightarrow W} x)) \tau^U$

$$(\lambda x^V. (P^{V \rightarrow W} x)) (f^{U \rightarrow V} \tau^U)$$

$f$ : optional term associated with either  $P$  or  $\tau$

$f$  **applies once to the argument** and not to the several occurrences of  $x$  in the function.

A conjunction yields

$$(\lambda x^V. (\wedge (P^{V \rightarrow W} x) (Q^{V \rightarrow W} x)) (f^{U \rightarrow V} \tau^U),$$

the argument is uniformly transformed.

Second order is not needed, the type  $V$  of the argument is known and it is always the same for every occurrence of  $x$ .

## D.10. FLEXIBLE vs. rigid use of optional morphisms

$(\lambda x^?. (\dots (P^{A \rightarrow X} x^?) \dots (Q^{B \rightarrow Y} x^?) \dots)) \tau^U$ :  
type clash(es) [Montague: ? = A = B e.g. e]

$(\Lambda \xi. \lambda f^{\xi \rightarrow A}. \lambda g^{\xi \rightarrow B}. (\dots (P^{A \rightarrow X} (f x^\xi)) \dots (Q^{B \rightarrow Y} (g x^\xi)) \dots))$   
 $\{U\} f^{U \rightarrow A} g^{U \rightarrow B} \tau^U$

$f, g$ : optional terms associated with either  $P$  or  $\tau$ .

For each occurrence of  $x$

with different  $A, B, \dots$  with different  $f, g, \dots$  each time.

Second order typing:

- 1) anticipates the yet unknown type of the argument
- 2) factorizes the different function types in the slots.

The types  $\{U\}$  and the associated morphism  $f$  are inferred from the original formula  $(\lambda x^V. (P^{V \rightarrow W} x)) \tau^U$ .

## D.11. Standard behaviour

$\phi$ : physical objects

*small stone*

$$\overbrace{(\lambda x^\phi. (\text{small}^{\phi \rightarrow \phi} x))}^{\text{small}} \overbrace{\tau^\phi}^{\text{stone}}$$

$$(\text{small } \tau)^\phi$$

## D.12. Qualia exploitation

*wondering, loving smile*

$$\overbrace{(\lambda x^P. (\text{and}^{t \rightarrow (t \rightarrow t)} (\text{wondering}^{P \rightarrow t} x) (\text{loving}^{P \rightarrow t} x)))}^{\text{wondering, loving}} \overbrace{\tau^S}^{\text{smile}} \\ (\lambda x^P. (\text{and}^{t \rightarrow (t \rightarrow t)} (\text{wondering}^{P \rightarrow t} x) (\text{loving}^{P \rightarrow t} x)))) (f_a^{S \rightarrow P} \tau^S) \\ (\text{and} (\text{loving} (f_a \tau)) (\text{loving} (f_a \tau)))$$

## D.13. Facets (dot-objects): incorrect copredication

Incorrect co-predication. The rigid constraint blocks the copredication e.g.  $f_g^{Fs \rightarrow Fd}$  cannot be **rigidly** used in

(??) *The tuna we had yesterday was lightning fast and delicious.*



## D.14. Facets, correct co-predication. Town example 1/3

$T$  town    $L$  location    $P$  people

$f_p^{T \rightarrow P}$     $f_l^{T \rightarrow L}$     $k^T$  København

*København is both a seaport and a cosmopolitan capital.*



## D.15. Facets, correct co-predication. Town example 2/3

Conjunction of  $\text{cospl}^{P \rightarrow t}$ ,  $\text{cap}^{T \rightarrow t}$  and  $\text{port}^{L \rightarrow t}$ , on  $k^T$

If  $T = P = L = e$ , (as in Montague)

$(\lambda x^e (\text{and}^{t \rightarrow (t \rightarrow t)} ((\text{and}^{t \rightarrow (t \rightarrow t)} (\text{cospl } x) (\text{cap } x)) (\text{port } x)))) k.$

Conjunction between three predicates... use **AND3**

$$\begin{aligned} & \Lambda \alpha \Lambda \beta \Lambda \gamma \\ & \quad \lambda P^{\alpha \rightarrow t} \lambda Q^{\beta \rightarrow t} \lambda R^{\gamma \rightarrow t} \\ & \quad \quad \Lambda \xi \lambda x^\xi \\ & \quad \quad \quad \lambda f^{\xi \rightarrow \alpha} \lambda g^{\xi \rightarrow \beta} \lambda h^{\xi \rightarrow \gamma}. \\ & \quad \quad \quad (\text{and}(\text{and}(P(f\ x))(Q(g\ x)))(R(h\ x))) \end{aligned}$$

$f$ ,  $g$  and  $h$  convert  $x$  to **different** types (flexible).

## D.16. Facets, correct co-predication. Town example 3/3

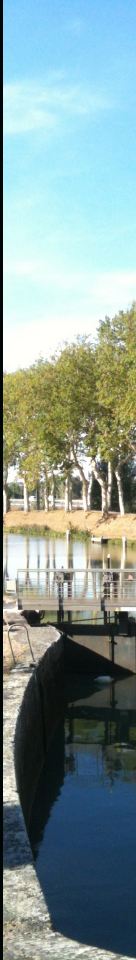
AND applied to  $P$  and  $T$  and  $L$  and to  $\text{cospl}^{P \rightarrow t}$  and  $\text{cap}^{T \rightarrow t}$  and  $\text{port}^{L \rightarrow t}$  yields:

$$\Lambda \xi \lambda x^\xi \lambda f^\xi \rightarrow \alpha \lambda g^\xi \rightarrow \beta \lambda h^\xi \rightarrow \gamma. \\ (\text{and}(\text{and}(\text{cospl}^{P \rightarrow t}(f_p \ x))(\text{cap}^{T \rightarrow t}(f_t \ x)))(\text{port}^{L \rightarrow t}(f_l \ x)))$$

We now wish to apply this to the type  $T$  and to the transformations provided by the lexicon. No type clash with  $\text{cap}^{T \rightarrow t}$ , hence  $\text{id}^{T \rightarrow T}$  works. For  $L$  and  $P$  we use the transformations  $f_p$  and  $f_l$ .

$$(\text{and}^{t \rightarrow (t \rightarrow t)} \\ (\text{and}^{t \rightarrow (t \rightarrow t)} \\ (\text{cospl}(f_p \ k^T)^P)^t)(\text{cap}(\text{id} \ k^T)^T)^t)(\text{port} \ (f_l \ k^T)^L)^t)^t$$





## **E The logical syntax of (generalised) quantification and generics**

## E.1. Why **type theory** for the syntax of semantics

opposed to Frege's single sort view:

$$\forall x:A P(x) \quad \equiv \quad \forall x. A(x) \rightarrow P(x)$$

(impossible for "most of")

in ancient and especially medieval philosophy (in particular Abu Barakat, Avicenna):

**we assert properties of things as being member of some class (= type?)**

There are less types than logical formulae with a single free variable, they are more constrained, and not any formula defines a comparison class.

## E.2. A personal view on the border between semantics and pragmatics

- semantics is encoded by the terms:  
they yield formulae by compositionality
- pragmatics is encoded in the types  
they are flexible and determined by the context



### E.3. Generic NPs

How do we logically formulate "most of" (much more than "the majority of") generic elements

- (1) The AKC notes that any dog may bite [...]
- (2) The Brits love France.
- (3) Un chien, ça peut toujours mordre.

idea to consider a fictive or fake element, like the  $\tau$  and  $\varepsilon$  of Hilbert like the  $\iota$  of van Heusinger.

(actually there is another reading for **2** that we are just starting to think about: Brits love France more than similar classes do (Germans, Italians, etc.))



## E.4. Radical minimalism / contextualism

Once we appeal to comparison classes (a type) and its generic element we can address the following puzzle issued from Frege's view of a single domain:

- My daughter is tall and thin for a 2 year old.
- My two-year-old can't get his own cup [...] because he can't reach, [...]

Carlotta who is a two year old girl it can be both tall and not tall, depending on her comparison class (her type in our type theoretic framework). Our type theoretical framework provides an account for such phenomena comparing Carlotta to the generic element of the corresponding class.



## E.5. Quantifiers in syntax

How do we derive formulae with generics from syntactic structures?

Syntax, natural or logical, is preferably finitely generated (otherwise, is it syntax?)

In usual Montague semantics, with a single individual type, first order quantification has type  $(\mathbf{e} \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$  as soon as we have a much richer type system, we would need a quantifier per type.

In F we have a **single** constant  $\forall$  of type

$\Pi\alpha. (\alpha \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$

It can be applied (specialised) to any type  $T$  to obtain the quantifier over the type  $T$ :

$\forall\{human\}(\lambda x^{human}.mortal^{human \rightarrow \mathbf{t}}(x)).$

## E.6. The syntax of "most" generic elements

Observe that there is a difference between most (which refers to a cognitively accessible class) and "most of the ... that ..." which refer to an arbitrarily set.

Consequently we shall have a first operator to obtain the generic from a type: **a constant  $\triangleleft$  of type  $\Pi\alpha. \alpha$**

When applied to a type  $T$ , this constant  $\triangleleft$  yields the element  $\triangleleft\{T\}$  of type  $T$  which is assumed to be the specimen of  $T$ :  $\triangleleft$  maps each type to its specimen.

As opposed to standard work, we do not say that the generalised quantifier is a property of two predicates: indeed we are in a typed version, and the restriction predicate of the usual setting is the type.

## E.7. The operator for "most of the ... that ..."

We need

**another constant  $\underline{\alpha}$  of type  $\Pi\alpha. (\alpha \rightarrow \mathbf{t}) \rightarrow \alpha$**

which creates a generic element corresponding to a property  
 $P : \tau \rightarrow \mathbf{t}$

But it is not an ordinary element of  $\alpha$  but an element of the subset defined by the property, say  $P$ , of type  $\tau \rightarrow \mathbf{t}$ . As for the definite article, one adds a presupposition,  $P(\underline{\alpha}\{U\}P)$

When using  $\lambda$ -DRT instead of plain  $\lambda$ -calculus there is a way for such property to percolate on top level using the  $\oplus$  of Muskens.



## E.8. Being tall (as a child) and not tall (as a human being)

We have some term and functions, with standard types: **Carlotta**  $Carlotta : 2yoGirl$  (constant) a class of child (these classes are vague)

$h : 2yoGirl \rightarrow human$  (**optional  $\lambda$ -term**) these classes are included in the *human* class.

We can thereafter say that she is tall if she is taller than the average element in her class, an interval, and the class can be modified according to the context.

But the important point is that we can state such things and that they participate without any problem to the compositional process.



## E.9. Being tall (as a child) and not tall (as a human being): computation

Here are the terms for:

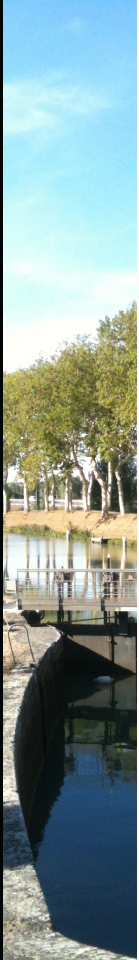
$$\text{height} : \Pi\alpha. (\alpha \rightarrow \text{float} \rightarrow \mathbf{t})$$
$$< : \text{float} \rightarrow \text{float} \rightarrow \mathbf{t}$$

The term for *tall* below says that it is higher than any of the heights of the specimen. That's a possible view, to turn functions into relations for such an element.

$$\mathbf{tall} \ \Lambda\alpha. \lambda x^{\alpha \forall \{\text{float}\}} \lambda h_s^{\text{float} \forall \{\text{float}\}} \lambda h^{\text{float}} \\ \text{height}\{\alpha\}(\angle\{\alpha\}, h_s) \wedge \text{height}\{\alpha\}(x, h) \Rightarrow h_s < h$$

type of tall:  $\Pi\alpha. \alpha \rightarrow \mathbf{t}$





## **F Other phenomena**



## F.1. Plurals

- (4) John and Mary sneezed. (= John sneezed and Mary sneezed, OK).
- (5) John and Mary met. ( $\neq$  \*John met and Mary met, OK).
- (6) The students wrote a paper. (“covering” reading: each student was part of group which wrote a paper)
- (7) Each student wrote a paper. (no “covering” reading, OK)
- (8) Three committees met. (two readings) OK
- (9) (?) The committee sneezed (coercion?)

With operators (similar to the treatment of generalised quantification)

## F.2. Virtual traveller (fictive motion)

(10) The path descended abruptly.

(11) The path descends for two hours.

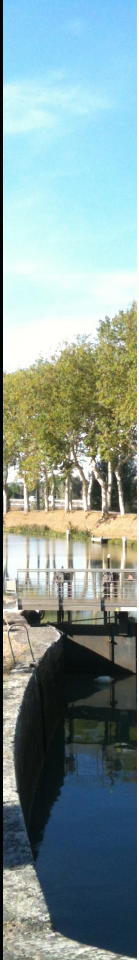
(12) The tarred road runs along the coast for two hours.

With transformation both from the verb and for the path...  
be careful for the virtual traveller not to be tarred! There are  
trickier cases:

(13) The fence zigzags from the plateau to the valley.

(14) The highway crawls through the city.





## **G Conclusion & prospects**

## G.1. What we have seen so far

A **general framework** for

the logical syntax of **compositional semantics**  
some lexical **semantics phenomena**

**Guidelines:**

**Terms: semantics**, sense, instructions for computing references

**Types: pragmatics**, defined from the context

**Idiosyncratic** (language specific) transformations compatible with the types but not forced by the types.

J'ai crevé. / ??? I went flat?

Ma voiture est crevée (roue) / ??? bouchée (injecteur)

**Practically: implemented** in Grail, Moot's wide coverage categorial parser, for fragment with a hand-typed semantic lexicon — but with  $\lambda$ -DRT instead of HOL in lambda calculus.



## G.2. Perspective 1: base types, relations & their acquisition

What are the base types?

How can they be acquired?

Can the optional modifiers be acquired,  
at least the specialisation modifiers?

Lexical data base Jeux de mots (M. Lafourcade).



### G.3. Perspective 2

What would be an adequate notion of subtyping? (for a systematic coding of the ontological specialisation relations that are often admissible in the language).

What about categorical interpretations (e.g. coherence spaces) as ontologies? (ontology-related question)



## G.4. Perspective 3: formulae vs. types

Typing ( $\sim$  presupposition) is irrefutable  $sleeps(x : cat)$

Type to Formula: a type  $cat$  can be mirrored as a formula that can be refuted  $\underline{cat} : e \rightarrow t \quad \underline{cat}(x) : t$

Formula to Type? Is any formula with a single free variable a type?  $cat(x) \wedge belong(x, john) \wedge sleeps(x) : type?$

At least it is not an implicit comparison class.



## G.5. Perspective 4: quantitative and compositional semantics

Our model integrates some lexical notions in compositional semantics.

Can we say something about the connection between vector semantics and compositional semantics?

Usually:

vector semantics: the topic that a discourse speaks about.

compositional semantics: which propositions are asserted.

Link with the work by A. Preller, V. Prince, et al.?

## G.6. Some references

### Work presented in this talk:

Montague semantics: chapter 3 of the *Logic of categorial grammars*, Springer 2012 (R. Moot, Ch. Retoré)

A lexicon for compositional semantics and lexical pragmatics: [article in the Journal of Logic, Language and Information, 2010](#) (Ch. Bassac, B. Mery, Ch. Retoré)

Fictive motion, virtual traveller [in French with plain Montagovian  \$\lambda\$ -terms \(TALN 2011\)](#) or [in English with  \$\lambda\$ -DRT \(CID 2011\)](#) (R. Moot, L. Prévot, Ch. Retoré)

Quantification: ["most" in this setting \(article in RLV\)](#) (Ch. Retoré)

Plurals: [a talk at the Coconat workshop](#) (R. Moot, Ch. Retoré)

### Related work:

Nicholas Asher *Lexical meaning in context: a web of words*. Cambridge University Press 2011

Zhaohui Luo *Contextual Analysis of Word Meanings in Type-Theoretical Semantics in Logical Aspects of Computational Linguistics 2012*, Springer LNCS 6736 edited by S. Pogodalla and J-Ph. Prost.