Some type theoretic accounts

of natural language semantics

Christian Retoré

Université de Bordeaux & IRIT (en 2012–2013)

Séminaire du thème 4 — IRIT, 5 octobre 2012

A Foreword – contents

A.1. 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

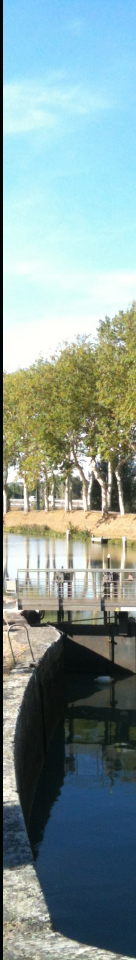 second order typed lambda calculus
 modelling lexical pragmatics
 a language for quantification, plurals,

Questions:

 base types? acquisition
 multi sorted logic vs. types theory
 sub-typing (specialisation relations)

B   Logical formulae as simply typed lambda terms and compositional semantics

B.1. 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.2. Types and terms: Curry-Howard

	Introduction rules	Elimination rules
Implication	$A\Gamma[A]_\alpha \Delta[A]_\alpha$ $\vdots$ $\dfrac{B}{A \to B}\ \to_i$ *some As are cancelled.*	$\dfrac{A \quad A \to B}{B}\ \to_e$

We shall focus on the $\to$ connective.

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

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

B.3. Terms

Types are.... formulae.

λ-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. $\to$ introduction if $x : U$ is a **variable** and $t : T$ then $(\lambda x^U. t) : U \to V$.

4. $\to$ elimination if $f : U \to V$ and $t : U$ then $f(t) : V$

With such typed terms we can faithfully encode proofs.

Variables are hypotheses (that are simultaneously cancelled).

B.4. 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 / λ-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.5. 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.6. Representing multi sorted formulae within lambda calculus — connectives

Assume that the base types are
   $\mathbf{e}_i$ (sorts for individuals, often there is just one) and
   $\mathbf{t}$ (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 $\mathbf{t} \rightarrow \mathbf{t}$ (negation)

- $\supset, \&, +$ of type $\mathbf{t} \rightarrow (\mathbf{t} \rightarrow \mathbf{t})$ (implication, conjunction, disjunction)

- for each i two constants $\forall$ and $\exists$ of type $(\mathbf{e}_i \rightarrow \mathbf{t}) \rightarrow \mathbf{t}$

B.7. Representing formulae within lambda calculus — language constants

The language constants for multi sorted First Order Logic:

- R_q of type $\mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (.... \to \mathbf{e}_{q_{s_q}} \to \mathbf{t}))$

  e.g. likes: $e \to e \to t$ sleeps $e \to t$

- f_q of type $\mathbf{e}_{q_1} \to (\mathbf{e}_{q_2} \to (.... \to \mathbf{e}_{q_{s_q}} \to \mathbf{e}_{q_0}))$

B.8. From formulae to normal lambda terms

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

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

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

B.9. Extension: higher order logic...

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

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

w has any property that an idiot has.

$$\forall P(\exists x \; stupid(x) \; and \; P(x)) \supset P(w)$$

Exercice: write it in lambda calculus.

B.10. 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 $M : A \to B$ 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.11. 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 $A \to B$ its is a lambda abstraction. (Sub)terms like $\forall sleep^{e \to t}$ are avoided.

This is achieved by replacing

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

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

B.12. The shape of normal lambda terms

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

Possibly $n = 0$ no λ 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** λ-terms

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

B.13. Normal terms of type t are formulae

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

1. normal λ-terms of type $\mathbf{e}_i$ with $x_k : \mathbf{e}_j$ as only free variables are logical terms with the same free variables

2. normal λ-terms of type $\mathbf{t}$ with $x_i : \mathbf{e}$ as only free variables are logical formulae with the same free variables and bound variables.

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

B.14.   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.15. Back to the roots: Montague semantics. Types.

Simply typed lambda terms

$$types ::= e \mid t \mid types \rightarrow types$$

chair , *sleep* $e \rightarrow t$

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

B.16. 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^*	$=$	$e \to t$	a noun is a subset of the set of entities
$(A \backslash B)^* = (B/A)^*$	$=$	$A \to B$	extends easily to all syntactic categories of a Categorial Grammar e.g. a Lambek CG

B.17. 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.18. Back to the roots: Montague semantics. Logic within lambda-calculus 2/2.

Words in the lexicon need constants for their denotation:

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

1. Replace in the lambda-term issued from the syntax the words by the corresponding term of the lexicon.

2. Reduce the resulting λ-term of type t its normal form corresponds to a formula, the "meaning".

word	**semantic type** u^*
	semantics : λ**-term of type** u^*
	x_v the variable or constant x is of type v
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	$e \rightarrow t$
	$\lambda x_e(\mathtt{statement}_{e \rightarrow t}\, x)$
speak_about	$e \rightarrow (e \rightarrow t)$
	$\lambda y_e\, \lambda x_e\, ((\mathtt{speak\_about}_{e \rightarrow (e \rightarrow t)}\, x)y)$
themselves	$(e \rightarrow (e \rightarrow t)) \rightarrow (e \rightarrow t)$
	$\lambda P_{e \rightarrow (e \rightarrow t)}\, \lambda x_e\, ((P\, x)x)$

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

((some statements) (themsleves speak_about)) of type t

$$\Big(\big(\lambda P_{e\to t} \, \lambda Q_{e\to t} \, (\exists_{(e\to t)\to t} \, (\lambda x_e(\wedge (P\ x)(Q\ x)))) \big)$$
$$\big(\lambda x_e(\mathtt{statement}_{e\to t} \, x) \big) \Big)$$
$$\Big(\big(\lambda P_{e\to (e\to t)} \, \lambda x_e \, ((P\ x)x) \big)$$
$$\big(\lambda y_e \, \lambda x_e \, ((\mathtt{speak\_about}_{e\to (e\to t)} \, x)y) \big) \Big)$$

$$\downarrow \beta$$
$$\big(\lambda Q_{e\to t} \, (\exists_{(e\to t)\to t} \, (\lambda x_e(\wedge_{t\to (t\to t)}(\mathtt{statement}_{e\to t} \, x)(Q\ x)))) \big)$$
$$\big(\lambda x_e \, ((\mathtt{speak\_about}_{e\to (e\to t)} \, x)x) \big)$$

$$\downarrow \beta$$
$$\big(\exists_{(e\to t)\to t} \, (\lambda x_e(\wedge (\mathtt{statement}_{e\to t} \, x)((\mathtt{speak\_about}_{e\to (e\to t)} \, x)x)))) \big)$$

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

$$\exists x : e \; (\texttt{statement}(x) \; \wedge \; \texttt{speak\_about}(x, x))$$

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).
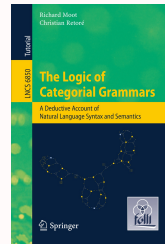
B.24. Good overall architecture / limits

Good trick (Church): a propositional logic for meaning assembly (proofs/λ-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 $\mathbf{e}_i$ and $\mathbf{t}$, as well as any type variable $\alpha, \beta, ...$ in P, are types.

- Whenever T is a type and α 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 $f : T \to U$.

- $\lambda x^T.\ \tau$ is a term of type $T \to U$ 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 α is a type variable, and $\tau : T$ without any free occurrence of the type variable α. (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 α, 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 α 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 stronly 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
$\Pi\alpha.\ (A \rightarrow B \rightarrow \alpha) \rightarrow \alpha$

Sum $A \vee B$ can be defined as
$\Pi\alpha.\ (A \rightarrow \alpha) \rightarrow (B \rightarrow \alpha) \rightarrow \alpha$

Existential quantification:
$\Sigma\beta.\ ((\Pi\alpha.\ (V[X] \rightarrow \beta)) \rightarrow \beta)$

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

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

List of β objects. (the β 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 function that can be programmed are the ones that can be proved total in second order Heyting arithmetic, that for such issues as 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 $Y : \Pi\alpha.\ (\alpha \to \alpha) \to \alpha$.

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 $\bot = \Pi\alpha.\ \alpha$ (relies on the comprehension axiom for all formulae of 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 mutisorted 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 $\mathbf{e}_i$ (many sorts of entities)

$\mathbf{v}$ (for events who play a particular role in λ-DRT)

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

Given types α, β and γ
three predicates $P^{\alpha\to t}$, $Q^{\beta\to t}$, $R^{\gamma\to t}$,
     over entities of respective kinds α, β and γ
for any ξ with three morphisms from ξ to α, to β, and to γ

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

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

D.4. 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 (λ-DRT).

We advocate a system based on *optional modifiers*.

D.5.   The Terms: main / standard term

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

D.6. The Terms: Optional Morphisms

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

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

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

D.7.   A Complete Lexical Entry

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

$$\left(\text{Paris}^T, \frac{\lambda x^T.\, x^T}{\varnothing}, \frac{\lambda x^T.(f_L^{T \to L}\ x)}{\varnothing}, \frac{\lambda x^T.(f_P^{T \to P}\ x)}{\varnothing}, \frac{\lambda x^T.(f_G^{T \to G}\ x)}{rigid} \right)$$

D.8.   RIGID vs flexible use of optional morphisms

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

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

f: optional term associated with either P or τ

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

A conjunction yields

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

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.9.   FLEXIBLE vs. rigid use of optional morphisms

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

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

f, g: optional terms associated with either P or τ.

For each occurrence of x

with different $A, B, ...$ with different $f, g, ...$ 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 $(\lambda x^V . (P^{V \to W} x)) \tau^U$.

D.10. Standard behaviour

ϕ: physical objects

small stone

$$\overbrace{(\lambda x^\varphi.\ \overbrace{(\text{small}^{\varphi\to\varphi}x)}^{\text{small}})}\ \overbrace{\tau^\varphi}^{\text{stone}}$$

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

D.11. Qualia exploitation

wondering, loving smile

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

$$(\lambda x^P. \ (\text{and}^{t \to (t \to t)} \ (\text{wondering}^{P \to t} \ x) \ (\text{loving}^{P \to t} \ x))))(f_a^{S \to P} \tau^S)$$

$$(\text{and} \ (\text{loving} \ (f_a \ \tau)) \ (\text{loving} \ (f_a \ \tau)))$$

D.12. 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.13. Facets, correct co-predication. Town example 1/3

T town L location P people

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

København is both a seaport and a cosmopolitan capital.

D.14. Facets, correct co-predication. Town example 2/3

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

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

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

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

$\Lambda \alpha \Lambda \beta \Lambda \gamma$
    $\lambda P^{\alpha \to t} \lambda Q^{\beta \to t} \lambda R^{\gamma \to t}$
        $\Lambda \xi \lambda x^\xi$
            $\lambda f^{\xi \to \alpha} \lambda g^{\xi \to \beta} \lambda h^{\xi \to \gamma}.$
                $(\text{and}(\text{and } (P \ (f \ x))(Q \ (g \ x)))(R \ (h \ x)))$
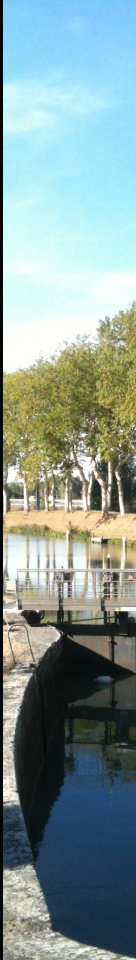
f, g and h convert x to **different** types (flexible).

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

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

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

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

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

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:

$$\forall x{:}A \; P(x) \qquad \equiv \qquad \forall x.\; A(x) \to P(x)$$

(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 of fake element, like the τ and ε of Hilbert like the ι of van Heusinger.

(actually there is an other 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 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 $(\mathbf{e} \to \mathbf{t}) \to \mathbf{t}$ 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

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

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

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

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

Observe that there is a difference between most (which refers to a cognitively accessible class) and "mots 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 $\measuredangle$ of type** $\Pi\alpha.\ \alpha$

When applied to a type T, this constant $\measuredangle$ yields the element $\measuredangle\{T\}$ of type T which is assumed to be the specimen of T: $\measuredangle$ 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 $\underline{\measuredangle}$ of type $\Pi\alpha.\,(\alpha \to \mathbf{t}) \to \alpha$

But it should be expressed at the same time that it is not an ordinary element of α but of the subset defined by the property of type $\alpha \to \mathbf{t}$.

In plain λ-calculus it is quite complicated, but in λ-DRT there are ways for such property to percolate on top level.

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* : 2*yoGirl* (constant) a class of child (these classes are vague)

   h : 2*yoGirl* → *human* (**optional λ-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:

$height : \Pi\alpha.\ (\alpha \to float \to \mathbf{t})$

$< : float \to float \to \mathbf{t}$

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

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

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

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.
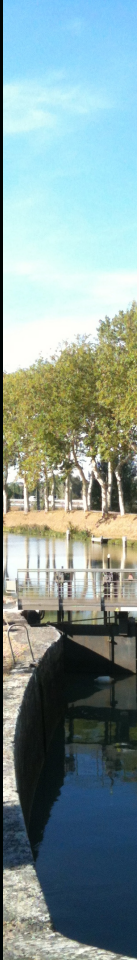
(11) The path descends for two hours.

(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

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? Je suis venu à pieds, mon vélo est crevé / *déraillé.

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

G.2. Perspective 1: base types, specialisation relations, subtyping

What are the base types?

How can they be acquired?

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

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

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

G.3.   Perspective 2: 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 $\underline{cat} : \mathbf{e} \to \mathbf{t}$ $\underline{cat}(x) : \mathbf{t}$

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.4. Some references

Work presented in this talk:

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 λ-terms (TALN 2011) or in English with λ-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 from Toulouse:

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

Sergei Soloviev and Zhaohui Luo Coercion Completion and Conservativity in Coercive Subtyping *Annals of Pure and Applied Logic* 113, 2000 and its application to lexical semantics 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.