Applying type theory and higher order logic on natural language syntax and semantics

Erkki Luuk (erkkil@gmail.com)

Department of Mathematics, Stockholm University, Sweden

Institute of Computer Science, University of Tartu, Estonia

(ver. 12th Nov 2014)

Logic

- Mathematical logic (in a general sense): a formal system of inference
- Expressiveness (an aspect of logic)
- Examples of logics sufficient to express the following statements:

  Propositional: P

  (contd. on the next slide)

Predicate logic

(contd. from the previous slide)

FOL: $\forall x\ Px$

SOL: $\forall P, x\ Px$

TOL: $\forall P_1, P_2, x\ P_1 P_2 x$

…

HOL: $\forall P_1, P_2, P_3, \ldots, x\ P_1 P_2 P_3 \ldots x$

Two remarks on expressiveness

- A *sentence* of predicate logic (of any order?) can be set as an atomic formula of propositional logic. However, the (possible) resulting propositional logic would be a metalogic, with all the substructure (the interpretations of $\forall$, $\exists$, P_n etc.) being lost
- Let P be a predicate *formula* of particular order. Then there is (in general) no lower order predicate formula Q s.t. $P \equiv Q$

Type theory

- Type: All terms (i.e. individuals, truth-values, functions or relations) in a logical system (e.g. nth-order logic) have a type
- Logical systems have relational or functional types; in most cases these are interdefinable (cf. Oppenheimer & Zalta (2011) for an argument that RTT is more general than FTT)

Type theory (2)

- Type (HOL) := a category associated w/ a term and identified by the order and arity of the latter (and by the arities of its arguments, of its arguments arguments etc. – a well-typed *n*-th order term must track the arities of its arguments to order 0? What if *n* is transfinite?)
- Type (TT) := a category of semantic value associated w/ a term

HOL and type theory

- HOL (1) (informal): a logic allowing predicating over predicates (i.e. TO and up)
- HOL (2): simple type theory
- Simple type theory: TT w/out dependent and polymorphic types. Ex: Russell's type theory, Church's type theory
- Modern (or “complex”) type theory. Ex: Martin-Löf type theory, Coquand's calculus of constructions

Lambek (1958)

- Lambek (1958), "The mathematics of sentence structure" (a variant of Categorial grammar and the earliest well-known example of applying TT on NL)
- Syntactic types ("parts of speech"):
  - primitive types s (sentence) and n (name)
  - compound types formed by the inductive definition: If x and y are types, then so are x/y ("x over y") and $y\backslash x$ ("y under x")

Lambek (1958) (2)

- Rewrite rules for concatenation: $(x/y)y \rightarrow x$

  $y(y\backslash x) \rightarrow x$

- An implicit "add matching parentheses" rule to group constituents (according to their "phrase structure" and to allow for the rewrite rules to operate)
- The formalism captures linear order (of concatenation) as well as subordinance and hierarchical constituent relations
- Ex: *John likes milk* : n n\s/n n
  - *John* (*likes* (*milk*)) : n(n\s/n(n)) → n(n\(s/(n))(n)) → n(n\s) → s
  - ((*John*) *likes*) *milk* : ((n)n\s/n)n → ((n)((n)\s)/n)n → (s/n)n → s

Lambek (1958) (3)

- Lambek's approach amounts to a description of NL predicate-argument structure w/ linear order (LPA – thus of NL syntax as well as sentential and phrasal semantics)
- Ex:

POS	Type	LPA
IV	n\s	$(x)P$
A	n/n	$P(x)$
CON	s\s/s	$(P)P(P)$

- Another component of his approach is a dedicated syntactic calculus (Lambek calculus – a formal language and deductive system primarily of interest to logicians)

Montague (1973)

- Montague ("The proper treatment of quantification...", 1973): *Syntactic* types ("categories" in the style of Categorial grammar):
  - Basic types: e (entity or individual expression) and t (declarative sentence)
  - Compound types: If *A* and *B* are types, then *A*/*B* and *A*//*B* are types (*A*/*B* and *A*//*B* play the same semantical but different syntactical roles)
  - E.g. IV phrases are of type t/e, T(erms – *John*, *Mary*, *he* etc.) of type t/IV, TV phrases of type IV/T

Montague (1973) (2)

- 17 syntactic rules, e.g.:
  - functional application: combining (concatenating) expressions of type IV/T and T yields one of type IV, combining t/t and t yields t etc.
  - rules for conjunction, quantification etc.
- *Semantics* is presented in terms of an *intensional logic* (a HOL). NL sentences are translated into the IL and analyzed in possible worlds semantics

Montague (1973) (3)

- Montague semantics (contd.):
  - 3 elementary types: the type of individuals e, type of truth values t∈{1,0}, type of indices (possible world - time pairs) s
  - 2 type-forming rules: 1. for any types a,b, $<a,b>$ is a type (the type of functions from a to b), 2. for any type a, $<\mathrm{s},a>$ is a type (an intensional type, the type of functions from indices to a)

Montague (1973) (4)

- Syntax-semantics (type) translation is given by the type-assignment function τ: $\tau(\mathrm{e}) = \mathrm{e}$, $\tau(\mathrm{t}) = \mathrm{t}$, $\tau(A/B) = \tau(A//B) = \langle\langle \mathrm{s},\tau(B)\rangle,\tau(A)\rangle$ (Bennett's (1974) simplification: $\tau(\mathrm{IV}) = \tau(\mathrm{CN}) = \langle \mathrm{e},\mathrm{t}\rangle$)
- An example translation of *John sleeps* into the IL (which (in the simplest case) would be sth like sleep(j)) goes as specified on the following 2 slides

John sleeps (Montague 1973)

- *John* : T = t/IV → <<s,τ(IV)>,τ(t)> = <<s,τ(t/e)>,t> → <<s,<e,t>>,t> (by Bennett) → $\lambda P.{}^{\vee}P$(john)
- Explanation: <s,<e,t>> := type of functions from indices to sets (i.e. properties) of individuals (prop^{i}); <<s,<e,t>>,t>, the type of functions from prop^{i} to {0,1}, is the type of properties of individual concepts (prop^{c}). Montague uses t-functional semantics, so <<s,<e,t>>,t> → $\lambda P.{}^{\vee}P$ (where P := "property", ${}^{\vee}X$:= the extension of X), i.e. a function taking prop^{i} as arg-s and returning (by λ-abstraction) their extensions (t-values). Finally λ-apply the function to argument john: $\lambda P.{}^{\vee}P$(john)

John sleeps (Montague 1973) (2)

- *sleep* : IV → <e,t> (by Bennett) → sleep′ (no translation rule for type <e,t> except for the generic $a \to a'$)
- Composition (translation rule T4): $F_4(a,b) \to a'(^\wedge b')$ ($^\wedge X$:= the intension of X): $\lambda P.{}^\vee P$(john) ($^\wedge$sleep′)
- β-conversion (λ-calculus): $\lambda P.[{}^\vee P$(john)]($^\wedge$sleep′) → $^{\vee\wedge}$sleep′(john)
- $^{\vee\wedge}$-elimination (Montague): $^{\vee\wedge}$sleep′(john) → sleep′(john)

Montague (1973) (7)

- Features of Montague grammar (1973):
  - Model-theoretic semantics
  - Truth-functionality and intensionality. Even PNs (*John* etc.) are of the type prop^{c} <<s,<e,t>>,t> rather than prop^{i} <s,<e,t>>, sets <e,t> or individuals e. "/…/ I regard the construction of /…/ [the] notion of truth under an arbitrary interpretation /…/ as the basic goal of serious syntax and semantics" (Montague 1970)

Montague (1973) (8)

- Features of Montague grammar (1973) (contd.):
  - Intensional logic. For tackling the meanings (i.e. truth-conditions) of words like *unicorn*, *seek* (we can seek nonexistent things) etc. In general, if *u* is a meaningful expression, then its intension is also a meaningful expression of type <s,*a*> (*a* the type of *u*)
  - Eclectic, idiosyncratic: categorial grammar in syntax; model theory, IL, HOL and λ-calculus in semantics
  - A fragment of English (e.g. no As; only quantified XPs in examples (XP := DP|NP) – what would it do w/ a S like *John likes milk*?)

Generalized quantifiers

- Mostowski (1957), Lindström (1966). Applications on NL: Barwise and Cooper (1981), ..., Westerståhl (2011), Keenan and Westerståhl (2011), etc.
- The idea (but not terminology) of NL applications due to Montague (1974, EFL): some XPs are **generalized quantifiers**
- Def: A generalized quantifier Q (of arbitrary type) is
- Syntactically, a variable-binding operator such that given a sequence of first-order formulas $\varphi_1,\ldots,\varphi_k$, $Q[x_1],\ldots,[x_k](\varphi_1, \ldots,\varphi_k)$ is a formula, and $Q[x_1],\ldots,[x_k]$ binds all free occurrences of $[x_1],\ldots,[x_k]$ in $\varphi_1,\ldots,\varphi_k$, resp. ($[x_i] := x_{i1},\ldots,x_{ini}$ for $1 \leq i \leq k$).

Generalized quantifiers (2)

- Semantically, a mapping from arbitrary universes (non-empty sets) *M* to a set Q_M of subsets of *M*, which interprets formulas of the form $Q[x_1],\ldots,[x_k](\varphi_1,\ldots,\varphi_k)$ according to the clause:
- $\mathbf{M} \vDash Q[x_1],\ldots,[x_k]\ (\psi_1([x_1],[b]),\ldots,\psi_k([x_k],[b]))$ iff $Q_M(\psi_1([x_1],[b])_{\mathbf{M},[x_1]},\ldots,\psi_k([x_k],[b])_{\mathbf{M},[x_k]})$

  where $\mathbf{M} = (M, I)$; $\psi_i([x_i],[y])$ a formula w/ $[x_i],[y]$ free; [b] a sequence of elements of *M* corresponding to [y]; $\psi_i([x_i],[b])_{\mathbf{M},[x_i]}$ the extension of $\psi_i([x_i],[y])$ in **M** relative to [b], i.e. the set of n_i-tuples $[a_i]$ s.t. $\mathbf{M} \vDash \psi_i([a_i],[b])$, where $[a_i]$ is a sequence of elements of *M* corresponding to $[x_i]$ (Mostowski 1957; Lindström 1966; Westerståhl 2014))

Generalized quantifiers (3)

- GQs (or just 'quantifiers') are second-order relations, so an *n*th-order quantifier (a maximal-order quantifier of nth-order logic) is an n+1*th*-order predicate
- GQs is thus an application of HOL (and of a proper subsystem of complex TT) on NL. Remark: there is (at least) one application of GQs using dependent types (Grudzinska and Zawadowski 2014)
- Ex-s: *a tall man* (linguistically, unquantified XP), *all men* (complex XP headed by a quantifier), *at least 8 but maybe less than a million men* (complex XP w/ at least 2 quantifiers)

Generalized Quantifier Theory

- In GQT sense, XPs are GQs. Linguistically speaking, not all XPs are GQs (e.g. *milk*, *horses*, *drunken men* etc.). Note that common nouns (*tree*, *milk* etc.) are not GQs, while all proper nouns (*John*, *Lake Ontario* etc.) are GQs. Also personal pronouns (*(s)he*, *him*, *their* etc.), demonstratives (*this*, *those* etc.) "determiners" (linguistically, determiners and quantifiers) (*a*, *the*†, *all*, *none*, *ten*, *at least 8* etc.) are GQs. As seen from their typing (next slide), GQs may include entire Ss in their scope

† The prevailing view in GQT

Generalized quantifiers: typing and beyond

- Relational typing: a GQ is of type $\langle n_1,\ldots,n_k\rangle$ ($n_i \geq 1$) iff it applies to k formulas and binds n_i variables in the i-th formula
- Examples (GQT; <...> type; each row's last type is syntactic, rest semantic; relational typing and the operational parts of GQs bold):
- <<s,<e,t>>,t> ~ **<1>** ~ <XP> {***John***, ***the linguist C. Woo***, ***this***, ***you***, ***her***, ***them***...}
- <<e,t>,<<s,<e,t>>,t>> ~ **<1,1>** ~ <CNP,XP> ~ $\mathbf{D_1}$+CNP → XP (D_1 := 1-place D (GQT)) {(***the*** | ***a***) *man*, ***all*** *poets slept*, ***more*** *grey* ***than*** *black rats* (*slept* | *mastered the rule*)...}
- <<<e,t>,<e,t>>,<<s,<e,t>>,t>> ~ <<1,1>,1> ~ **<1,1,1>** ~ <<CNP,CNP>,XP> ~ $\mathbf{D_2}$+2CNP → XP {***more*** *rats* ***than*** *cats* (*slept* | *mastered the rule*)...}

Generalized quantifiers: typing and (way) beyond

- <<e,t>,<<e,t>,<e,t>>> ~ <1,<1,1>> ~ **<1,1,1>** ~ <CNP,<IVP,IVP>> ~ **D_1**+CNP+2IV → 2IVP {***more*** *rats slept* ***than*** *crept…*}
- <<<e,t>,<e,t>>,<<e,t>,<e,t>>> ~ <<1,1>,<1,1>> ~ **<1,1,1,1>** ~ <<CNP,CNP>,<IVP,IVP>> ~ **D_2**+2CNP+2IV → 2IVP {***more*** *rats slept* ***than*** *cats crept…*}
- <<<e,t>,<e,t>>,<<s,<<s,<e,t>>,t>>,<e,t>>> ~ <<1,1>,2> ~ **<1,1,2>** ~ <<CNP,CNP>,TVP> ~ **2D_1**+2CNP+TV → TVP {***more than seven*** *rats bit* ***four*** *cats…*}
- <<<e,t>,<<e,t>,<e,t>>,<e,t>>,<?>> ~ <<1,<1,1>,1>,3> ~ **<1,1,1,1,3>** ~ <<CNP,<CNP,CNP>,CNP>,DVP> ~ **2D_1+D_2**+4CNP+DV → DVP {***more than seven but probably less than a million*** *rats gave* ***more*** *roses* ***than*** *lilies to* ***at least 2*** *cats…*}

Generalized Quantifier Theory: features

- Interpreting XPs (in GQT sense) and larger NL structures (possibly w/ entire Ss in their scope) as GQs
- Handling of complex mono- and polyadic (pertaining to unary and binary/ternary Vs, resp.) allegedely quantificational phenomena in NL
- Handling intensions as well as extensions
- Disambiguating scopes and readings and computing logical forms of certain NL expressions

MG and GQT: shortcomings

- By default, uninterested in / do not adequately account for:
  - Anaphora and other "dynamic" phenomena (and interface(s) to morphosyntax in general)
  - Sufficiently fine-grained semantic typing (e.g. Luo 2010, Asher 2014)
  - Typological diversity of human language
  - Cognitive/psychological plausibility of its models and interpretations
  - In silico implementability of the formalisms and results
  - Developing useful frameworks or formalisms for descriptive, applied or computational linguistics

MG and GQT: impact

- For many logicans and (analytic) philosophers of language:
  - The legacy and bread-and-butter work in theoretical formal semantics of NL
- For most linguists:
  - Definitions and terminology incompatible w/ linguistics
  - The role of quantification in NL blown out of proportion (both in principle and wrt. its applicability to and scope in particular NL expressions)
  - Disjoint from (and difficult to reconcile w/) linguistics
- In general:
  - GQT (1981-...), primarily notable for its interpretation of NL quantification, is a direct continuation and significant extension of MG (1970-1974)
  - For (largely) historical reasons, MG-GQT is probably the leading branch of theoretical formal semantics of NL

Ranta (1994)

- Ranta ("Type-theoretical grammar", 1994), a framework for analyzing NL syntax and semantics based on Martin-Löf (or intuitionistic or constructive) type theory
- Propositions as types principle (MLTT): propositions are sets, proofs (specifically, proof objects) are elements. The truth of a proposition means that the set has an element. E.g. the proposition $A\&B$ is true (i.e. proven) by the set $\{\{P,Q\}\}$, where P is a proof object of A and Q a proof object of B

Proof (in Martin-Löf type theory)

- Proof object vs. proof process (MLTT):

  1. $x : A$

  …

  <u>$n.\ b(x) : B$</u>

  n+1. $\lambda x.b(x) : A \to B$

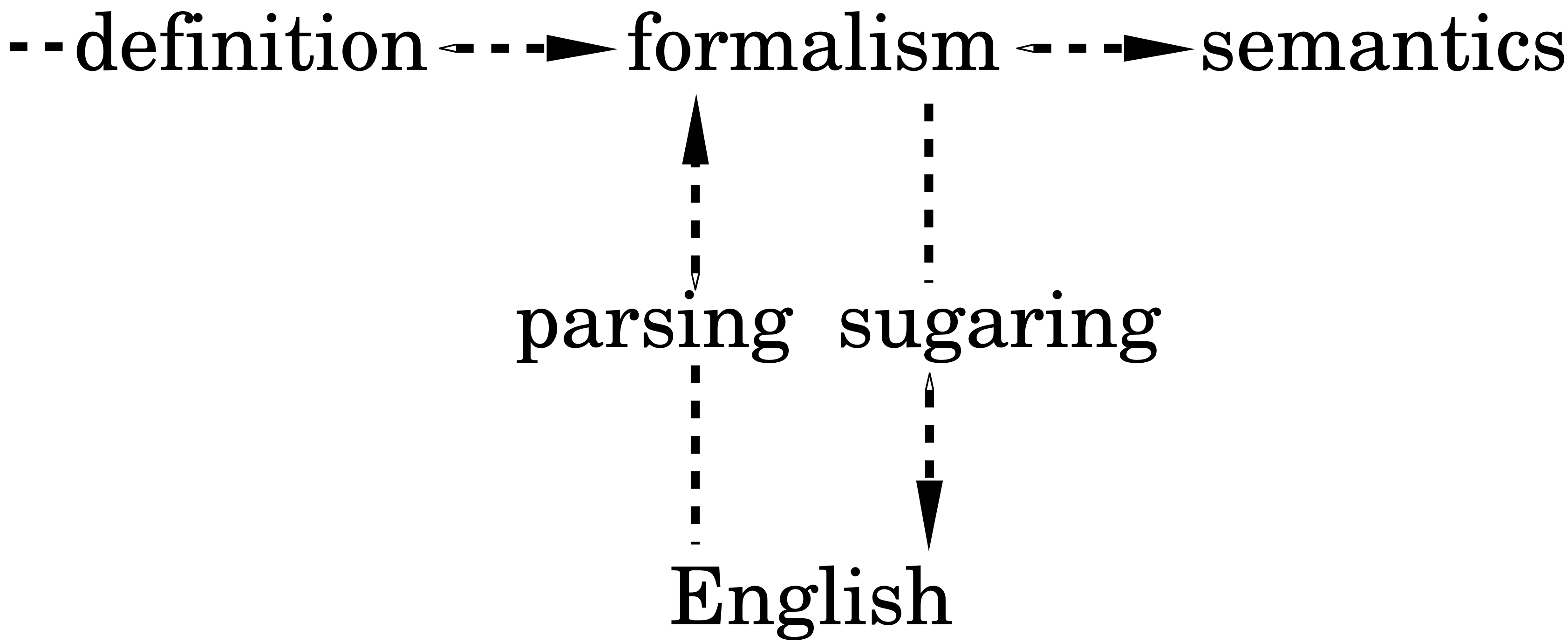
  Proof object is $\lambda x.b(x)$, proof process is the sequence of rows (1, …, n+1) (":" := "is an element of" ≡ "is of type"). In this case, the proof object $\lambda x.b(x)$ ≡ the set of pairs $(x,b(x))$ in the function ≡ the pair of rows (1, n)

Back to Ranta's TTG (1994)

- The kind of semantics implemented by MLTT, TTG and other similar frameworks is called *proof-theoretic* (and contrasted with model-theoretic semantics)
- TTG represents NL syntax and semantics on a single level
- NL generation is divided into 2 components: defining grammatical representations ("formalism" or "parse trees") and sugaring (transforming the unambiguous "formalism" to potentially ambiguous (but "readable") strings)

Ranta (1994) (3)

- TTG (the general picture):



- The path (definition, formalism, sugaring, English) is generation

Ranta (1994) (4)

- Ex: the full formalization of *a man walks* (the proof process of the corresponding proof object ("($x : man$)" is the premise and "1." the label of the hypothesis immediately below them)):

$$
\dfrac{\dfrac{\begin{array}{cc} & (x : man) \quad 1. \end{array}}{}\qquad}{}
$$

$$
\dfrac{man : \text{set} \qquad \dfrac{x\ walks : \text{proposition} \qquad x : man}{x\ walks : \text{proposition}}\ subst.}{(\Sigma x : man)(x\ walks) : \text{proposition}}\ \Sigma F, 1.
$$

Type theory w/ records

- **Type theory w/ records** (e.g. Cooper 2005) is another proof-theoretic approach to NL semantics and syntax based on MLTT
- Capitalizes on dependent types (a feature of MLTT and other modern TTs):

  $A(a_1, \ldots, a_n)$:= type A depending on objects a_1, …, a_n

Type theory w/ records (2)

- If $a_1 : T_1$, $a_2 : T_2(a_1)$, …, $a_n : T_n(a_1, …, a_{n\text{-}1})$, a record $[l_1 = a_1, …, l_n = a_n, …]$ is of type $[l_1 : T_1, l_2 : T_2(l_1), …, l_n : T_n(l_1, …, l_{n\text{-}1})]$. Thus a record type is a set of fields consisting of a label and type (Cooper 2005)
- Ex: *a man walks* corresponds to a record type $[x : Ind$, $c_1 : \text{man}(x), …, \text{c}_2 : \text{walk}(x)]$. A record of this type is $[x = a, c_1 = p_1, c_2 = p_2]$, where a : *Ind* (the type of individuals) and p_1, p_2 are proofs of man(a) and walk(a), resp. Note that the record may have had additional fields and still be of this type. The types man(x), walk(x) are dependent types of proofs

Subtyping

- Pervasive in NL, e.g.:
  - [spruce] ≤ [tree] ≤ [plant] ≤ P (P := physical object)
  - [large book] ≤ [book]
- Contravariant propagation of subtyping for function types (Reynolds 1981):

$$\frac{A \leq A' \qquad B \leq B'}{A' \to B \ \leq \ A \to B'}$$

  - E.g. since [John Smith] ≤ [John] and [famous man] ≤ [man],

    [John] → [man] → PROP < [John Smith] → [famous man] → PROP

    [John is a man] < [John Smith is a famous man] (PROP := proposition)

Subtyping (2)

- "Subsumptive" subtyping:

$$\frac{a : A \qquad A \leq B}{a : B}$$

- The problem: "subsumptive" subtyping introduces new objects into a type, which is incompatible w/
  - Canonicity: Any closed object of an inductive type is definitionally equal to a canonical object of that type
- Solution: **Coercive subtyping** (Luo 1999-…):

$$\frac{\Gamma \vdash f : (B)C \qquad \Gamma \vdash a : A \qquad \Gamma \vdash A <_c B : Type}{\Gamma \vdash f(a) = f(c(a)) : C}$$

Subtyping (3)

- Rules that extend coercive subtyping to local contexts, allowing for interpretations of sentences like *omelette wants the bill* etc. Since
- [want] : [animate] → E → PROP   (E := entity)
- [omelette] < [inanimate]

coercions are required (and can be introduced – Luo 2010) for local contexts, allowing for [omelette] < [animate] and the expression to be well-typed in appropriate contexts

Fine-grained typing in MTT: copredication

- MTT allows for straightforward accounts of **copredication**, as in *J picked up and mastered the book*, the well-typedness of which is ensured by
- [pick up] : [human] → P → PROP

  < [human] → P&I → PROP

  < [human] → [book] → PROP
- [master] : [human] → I → PROP

  < [human] → P&I → PROP

  < [human] → [book] → PROP

  (I := informational object; P&I < P; P&I < I)
- MG/GQT interpretation of copredication is usually much more complex

Fine-grained typing in MTT: selectional restrictions

- Differently from MG and GQT, MTTs allow for fine-grained typing of concepts (Luo 2010, Asher 2014):
- MG/GQT: CNP, IVP : <e,t>
- MTT: [man], [human], [spruce] : *Type*
- MG/GQT cannot account for **selectional restrictions**:
  - MG/GQT: [talk] : <e,t>
  - MTT: [talk] : [human] → PROP
- Differently from MG/GQT, MTT can account for type clashes in NL expressions (e.g. *a table talks*, *green ideas* etc.)

Fine-grained typing in MTT: two-levelled semantics

- MTT accommodates 2 kinds of semantics: those of presupposed and proffered types (Asher 2014)
- Allows for logical forms w/ presupposed types for type-checking, e.g.

  $\lambda P{:}_{\mathrm{P \to PROP}}\, \lambda x{:}_{\mathrm{P}}\, (Px \wedge \mathrm{RED}x)$ for expressions _ *is red* or *red* _ (P := physical object, PROP := proposition)

- The eventual (proffered) types will be usually even more fine-grained, because

  $\mathrm{RED}(\alpha) \leq \alpha \leq \mathrm{P}$, w/ α the proffered type

Category theory and NL

- Lambek (1988) “Categorial and Categorical Grammars”, de Groote (2001), Pollard (2011), Asher (2014), Preller (2014)
- Metatheory (except for Preller 2014): setting up a categorical framework for a linguistic (esp. semantic) theory (mostly very general descriptions of NL using CCCs, Topos, a pre-Boolean algebra object PROP, Stone duality, biproduct dagger categories…)

Conclusions (if any)

..and thanks

Meanwhile in a single-sorted HOL

- $A(B(C(\boldsymbol{x})))$
- $A'(B'(C'(\boldsymbol{y})))$

where $\boldsymbol{x}$,$\boldsymbol{y}$ are m,n-tuples of individuals, resp.