EasyChair Preprint

№ 1516

A type-theoretical reduction of morphological, syntactic and semantic compositionality to a single level of description

Erkki Luuk

September 25, 2019

A Type-Theoretical Reduction of Morphological, Syntactic and Semantic Compositionality to a Single Level of Description*

Erkki Luuk
Institute of Computer Science, University of Tartu, J. Liivi 2, Tartu 50409, Estonia
erkkil@gmail.com

Abstract

The paper presents NLC, a new formalism for modeling natural language (NL) compositionality. NLC is a functional type system (i.e. one based on mathematical functions and their types). Its main features include a close correspondence with NL and an integrated modeling of morphological, syntactic and semantic compositionality. The paper also presents an implementation of NLC in Coq. The implementation formalizes a diverse fragment of NL, with NLC expressions type checking and failing to type check in exactly the same ways that NL expressions pass and fail their acceptability tests. Among other things, this demonstrates the possibility of reducing morphological, syntactic and semantic compositionality to a single level of description. The level is tentatively identified with semantic compositionality — an interpretation which, besides being supported by results from language processing, has interesting implications on NL structure and modeling.

1 Introduction

As shown by Asher (2014), Luo (2010, 2014) and Ranta (1994), in a logical approach (i.e. in one with simpler alternatives such as zeroth, first, second and higher order logic), complex type theories outshine simpler ones in accounting for natural language (NL) phenomena like anaphora, selectional restrictions, etc. The basic judgement in type theory, $a : A$, is read "term a has type A", where

(i) type := a category of semantic value.

Since the notion of type is inherently semantic, it is by definition suited for analyzing universal phenomena in NL, as NL semantics is largely universal (as witnessed by the possibility of translation from any human language to another).

2 Interpreting Natural Language

NL functions and function applications have a distinctive form. Specifically, any *morphosyntactically admissible concatenation* $\mathsf{c}(a, e_1, ..., e_m)$, which is parsed as $a(e_1, ..., e_m)$, is a function or function application term (FFAT) in NL ($e_1, ..., e_m$ may be null). For example, *red*, *red book*, *livre rouge*, etc., are FFATs in idiosyncratic notations (viz. English, French, etc.). As linguistic expressions are FFATs, they are naturally parsed as functions or function applications.

How to decide whether a particular application $a(e_1, ..., e_m)$ holds? Usually, one has a basic intuition about what modifies what (modification is a subcase of function). The main sources for the intuition are morpheme or word classes and semantic contribution tests. For example, *-s* modifies (i.e. is a relation over) *work* in *works* rather than vice versa, as (1) affixes modify stems not vice versa, (2) person/tense and plural markers modify flexibles[1] rather than vice versa, and (3) *-s* contributes to the meaning of *work* in *works* rather than vice versa. By a similar argument, *heavy* modifies *rain* in *heavy rain* rather than vice versa, *sleeps* modifies *john* in *john sleeps* rather than vice versa, etc. In each case, there's a clear asymmetry between functions of the components, as conveyed by (1)–(2) the functions of word and morpheme classes and (3) semantic contribution tests.

While the above methods may seem sufficient by normal standards, for type-driven theories an even more rigorous way of testing NL relations

*Accepted to RANLP 2019 (http://lml.bas.bg/ranlp2019).

[1] See Luuk (2010).

is available. We can model a (reasonably large) fragment of NL in a suitable type system to make it *(fail to) type check in exactly the same ways as (hypothetical) NL expressions pass (and fail) acceptability tests*. Call this principle "the correspondence criterion". Arguably, this possibility, the ultimate test for any type-driven linguistic theory, has not been fully explored, while several significant steps in this direction have been taken. Chatzikyriakidis and Luo (2014b, 2016) describe (among other things) a use of proof assistants for testing the logical soundness of linguistic theories, while Grammatical Framework (GF, Ranta (2004)), a statically typed programming language for writing NL grammars, gets closest to a *type-theoretical implementation of NL*. However, GF is a high level tool, mathematically opaque to the end user, and quite specialized. Because it is geared towards writing NL grammars, it does not offer a selection of different mathematical formalisms to work with, being thus unsuitable for a general (low level) modeling of NL and theories thereof. In addition, it does not concern itself with modeling compositional semantics, although a FrameNet API for GF has been proposed and partly implemented (Gruzitis et al., 2012; Gruzitis and Dannélls, 2017).

A powerful feature of typed theories (especially of the richly typed[2] ones — Chatzikyriakidis and Luo (2014b)) is that they allow to capture not only grammatical but also semantic acceptability. The paper shows that a combination of functions and rich typing makes it possible to use *a single type system for modeling the core of NL morphology, syntax and compositional semantics*, thus questioning the soundness of the theoretical distinction between these different NL "layers" (and partly eradicating their even more theoretical "connections" such as the syntax-semantics interface). Because of a wide selection of mathematical formalisms in a richly typed setting, combined with a relatively "instant" compile time type checking, proof assistants (e.g. Coq, Agda, Lean) are suitable tools for such work (cf. Chatzikyriakidis and Luo (2016)). As shown below, implementing polymorphic functions with a subclass of compound types (called lump types) allows to partly collapse different "levels" of NL (morphology, syntax, and compositional semantics).

[2]Rich typing coincides mainly (although perhaps not exclusively) with dependent and/or polymorphic types.

3 A Note on Selectional Restrictions

More or less overlooked in Montagovian (Montague, 2002) and categorial (Lambek, 1958) traditions, selectional restrictions have recently become a focus of intense research in modern type theories (Asher, 2014; Luo, 2010; Bekki and Asher, 2013). The essence of the semantic (and perhaps even more logical than linguistic) phenomenon of selectional restrictions is prescribing types for a relation's arguments.

There is an important difference between (1) arguments belonging to types and (2) relations imposing types on their arguments. While an argument can clearly belong to different types[3], a relation should not impose different types on its kth argument, for a fixed k. Modeling selectional restrictions along with grammar is an important motivation for lump types, described below (in section 5.1.1).

4 Introducing NLC

Call the formalism we are considering NLC. It is a type system for modeling NL syntax, morphology and compositional semantics — briefly, compositionality in NL. The basic unit of NLC is a function of a small (usually ≤ 3) arity. The expressions of NLC are functions, function applications, function types (Π-types), lump types, terms of lump types, and universes (types of types). Elementary terms of NLC are functions. This is possible if we interpret "proper arguments" as nullary functions (functions that take no arguments). So functions are divided into proper arguments and proper functions (the latter being functions that take arguments). For a fixed NL k, let T^k be a proper type variable of NLC, where "proper type" refers to a type that is not a universe, and $\mathcal{M}^k$ the set of morphemes[4]. Then we have the rule:

$$\frac{a \in \mathcal{M}^k}{a : T^k}\ \text{ATV-Intro},$$

for generating atomic terms of NLC and introducing type variables for them. The rule ATV-Intro says that all morphemes have types in NLC (technically: "if a is a morpheme of language k then a has type T^k").

Some morphemes (e.g. stems) occur only in an argument position (i.e. are proper arguments),

[3]E.g. a book is a physical and informational object.

[4]Morphemes are smallest signs (form-meaning correspondences) in language.

while others (e.g. plural markers) are proper functions. More generally, a function or function application is a parsimonious interpretation of morphemes, words, phrases and sentences in NL. *Sentences, multimorphemic words and phrases are function applications*. This amounts to a rigorous interpretation of the more general "principle of compositionality", as it is known at least since Frege[5]. Broadly speaking, there are only two ways to explain the emergence of compositional meaning: by specifying a relation together with its (a) arguments or (b) type. The first corresponds to e.g. function application and the second to function declaration.

In a functional type system, generating complex terms and introducing type variables for them is straightforward (rule CTV-Intro, with T_i^k ranging over proper types):

$$\frac{e_1 : T_1^k, ..., e_m : T_m^k \quad a : T_1^k \to ... \to T_m^k \to T_{m+1}^k}{a(e_1, ..., e_m) : T_{m+1}^k}$$

where $a(e_1, ..., e_m)$ is an application and $T_1^k \to ... \to T_{m+1}^k$ the usual (right-associative) function type. Since CTV-Intro is the standard function type elimination rule, the function type introduction rule is derivable from CTV-Intro.

5 NLC: The Types

Since grammatical (and semantic) categories have a limited, finite number of members, we need some atomic types with limited membership. Let $\mathcal{U}$ denote the top-level universe of NLC. We use axioms of the form $S : \mathcal{U}$ and $T : S$, where S may be a universe, for introducing atomic type constants corresponding to linguistic categories like stem, case, nominative, noun, verb, etc. For proper functions, we need function types (Π-types). Besides this, we need only polymorphism and lump types, both of which can be (in various ways) implemented with function types.

Since a term of type A may contain another term of type A (or in case of a function type, take another term of the type as an argument), we have sufficient complexity without recursion (self-reference, which we do not need). For example, a sentence A containing another sentence B does not imply recursion unless $A = B$ or B references A. Thus, we have same-type-reference without self-reference.

5.1 Polymorphism and Lump Types

The complexity of NLC goes well beyond regular function types. In considering a NL expression type-theoretically, one is frequently inclined to assign it to more than one type. Confining our analysis to only the linguistically relevant features, we may want to type e.g. *stone* as a flexible, physical object, word in nominative case, etc. A way — corresponding to polymorphism — to go about this is to define coercions to (i.e. coercive subtyping for) all the types we need. In fact, we have three possibilities: either we (1) lose some type information, type *stone* (2) polymorphically or (3) with a lump type.

As the nominal and verbal readings of *stone* preclude each other, a polymorphism is required if we want to encode them both (cf. footnote 7). In many other cases, however, a lump type may be preferred. Thus, NLC features types for morphemes, function types, lump types and polymorphism.

5.1.1 Lump Types

While possibility (3) is new, the superclass of lump types, compound types have been used for NL modeling in the form of multi-field record types (Cooper, 2005; Ranta, 2004; Luo, 2011; Chatzikyriakidis and Luo, 2014a). Also, some kind of polymorphism (e.g. by subtyping — Luo (2010)) is frequently thought to be necessary. However, the use of compound types has been so far confined to record types only, i.e. not properly generalized[6]. A compound type is a type which is a syntactic compound of multiple types or their terms. Normally, the compounded types are different; in the degenerate case, they are the same. Examples (or implementations) of compound types include Σ-, Π-, Cartesian product and multi-field record types.

We defined types as "categories of semantic value" but, as the example of *stone* shows, for NL expressions the value covers not only linguistic semantics but also the meanings of syntactic and

[5]The principle is more general because it holds also for interpretations of formal languages.

[6]In many (most?) programming languages that support them, the notion of "compound type" (or "compound data type") is synonymous with a multi-field record type. This is *not* the way it is used here. While a record type can be defined as a (mathematically more fundamental) Σ-, Π- or Cartesian product type (Constable, 2003), I have never seen it defined as a function application.

morphological categories (we will return to this point in section 9). As compared to (1) and (2), packing an expression's meaning into a lump type allows to do away with both the loss of information and typing complexity. Of course, the lump type itself will be complex but this will, hopefully, present less problems than alternatives (1) and (2). As a bonus, the underlying linguistic model will simplify on account of reducing compositional semantics and parts of morphology and syntax to a single level of description. Below is the rule for lump type introduction (LT-Intro):

$$\frac{B : T^k \quad c_0 : C_0^k,\ ...,\ c_{n+1} : C_{n+1}^k \quad B \mapsto c_0, ..., B \mapsto c_{n+1}}{B : C_0^k..C_{n+1}^k}$$

where T^k is a proper type variable, B a term constant and $C_0^k, ..., C_{n+1}^k$ type constants in NLC, $x \mapsto y$ a function interpreting x as y[7], and $C_0^k..C_{n+1}^k$ the notation for a lump type (comprising types C_0^k through C_{n+1}^k, i.e. there must be at least two). LT-Intro is formalism-agnostic — the exact mathematical structure used for lumping is irrelevant. In particular, as shown in Supplement A[8], we can implement lump types for NL as 1) record types, 2) function applications, 3) Cartesian product types, or 4) Π-types. In languages that have them (e.g. TypeScript, Flow...), it is natural to encode lump types as intersection types. Lump types are defined as compound types that satisfy LT-Intro (i.e. we are not interested in empty lump types).

Supplement B[9] proceeds to formalize a diverse fragment of NL with function applications. The fragment comprises stems, nouns, verbs, flexibles, proper names, pronouns, XPs[10], adjectives, sentential, adjectival and generic adverbs, determiners, demonstratives, quantifiers, tense-aspect-mood, gender, number and nonfinite markers, cases, adpositions, sentences (both simple and complex), connectives, connective phrases (for substantives, adjectives, adverbs and sentences), complementizers, copulas, and selectional restrictions (for physical, informational, limbed, biological, animate and sentient entities).

[7]The interpretations must not preclude each other; if they do (as e.g. the interpretations of *run* as a noun and verb), they are dealt with polymorphism instead.

[8]https://gitlab.com/jaam00/nlc/blob/master/compound.v

[9]https://gitlab.com/jaam00/nlc

[10]Frequently (and theory-dependently) alternatively referred to as NPs or DPs.

The linguistic categories not formalized in Supplement B are gerunds, participles, auxiliary verbs, interrogatives, numerals, negation, mass/-count distinction and unspecified selectional restrictions (and possibly others). These are omitted not because of a special difficulty formalizing them would pose but because the formalized fragment is sufficiently expressive (and representative of NL) to make the points of utility and feasibility of NL formalization with the combination of compound and polymorphic types. The formalization has been done in the proof assistant Coq (ver. 8.9), making use of its features like Ltac programming, custom notations, etc. Besides showing the use of lump (viz. application) types in NL modeling, Supplement B should fulfill the abovementioned "correspondence criterion".

5.1.2 Polymorphism

Besides lump types, there is some use for polymorphism as well — if not for any other reason, then because NL expressions may be underspecified. E.g. *sleep* and *stone* are flexible stems that can function both as nouns and verbs. As a verb, *sleep* selects for a specific argument, say, a sentient entity (only higher animals can sleep — for trees, stones and bacteria it is not an option). As a noun, it is quite similar to many others: a stem, a flexible in singular, an event, etc. Since verbs are functions, *sleep*'s type must be a function type, but since it also functions as a noun, a polymorphism is desirable. The alternative, defining two distinct *sleeps*, one noun and one verb, would be redundant and inelegant — esp. in a programming language, where (differently from NL) they would have to be formally distinct (e.g. `sleep` and `sleep0`) even in the absence of any discriminating context (markers and/or arguments).

There are several ways to implement polymorphism, but dependent and/or polymorphic types and subtyping are the most common. For example, Coq supports polymorphic types (e.g. `∀x:Type, x`), but an additional formalization layer is sometimes desirable to improve type inference. One of the main obstacles for formalizing NL in Coq (and likely also in other proof assistants) is that NL type inference is much more powerful than that of the (terms of) relatively simple types like sorts, sensu stricto variables (e.g. those defined with `Parameter` and `Variable` in Coq) and function types over them. The reason is that the simple types have an unbounded num-

ber of terms, while in NL the number is fixed, very limited, and usually known in advance[11]. The only counterexamples to this rule are phrases, clauses and complex words. So the additional layer of formalization is used for downgrading the overpowerful simple types to something on which type inference would work. In Coq, the main devices for such downgrading are inductive types, "canonical structures", and type classes. Our formalization uses them all, relying most heavily on canonical structures (essentially, canonical records of a record type).

6 Truth-Functionality

So far, the semantics developed here is not truth-functional, i.e. it is type- but not model-theoretic. As type theory is 'semantic' by definition, it is clearly sufficient for a semantical cast of semantics without a recourse to model theory. This is evident in programming language semantics, where the role of model theory is marginal as compared to that of type theory. Traditionally, in natural language semantics the opposite is true, as sentential semantics is usually construed as model-theoretic even in a type-theoretical setting (e.g. Chatzikyriakidis and Luo (2015, 2016)). The obvious reason for this is that the (prevailing, i.e. Montagovian) tradition is model-theoretic. For this reason, an optional truth-functionality module has been added to the implementation. Degenerate models (where all NLC sentences (S) are uniformly true, false or undecidable) can be specified trivially by subtyping, e.g.

```
Parameter s_prop:> S -> Prop.
(* all S-s undecidable *)
```

and with only a little effort non-trivial models can be specified, too (with subtyping and a special notation matching NLC constructions with appropriate values). Below is an example from Supplement B:

```
Check $(PRES walk john): Prop. (*False*)
Fail Check $(PAST walk stone). (*type mismatch*)
Check $(PRES sleep john). (*True*)
Check $(PRES sleep (PL boy)). (*False*)
Check $(PAST sleep (-s boy)). (*True*)
Fail Check $sleep. (*type inference fail*)

(*a trivial proof that "boys
don't sleep" and "john sleeps"*)
Theorem pres: (~ ($ (PRES sleep (-s boy)))) /\
($ (PRES sleep john)). Proof. firstorder. Qed.
```

[11] The latter will depend on your linguistic theory, as different theories posit different categories and members for them.

7 Comparisons

In section 2 we compared NLC with related typed approaches. This section takes a broader (albeit still related) perspective, comparing NLC with Combinatory Categorial Grammar and Head-Driven Phrase Structure Grammar.

7.1 Combinatory Categorial Grammar

A feature of NLC, Combinatory Categorial Grammar (CCG) (Steedman, 2000) and some other categorial formalisms is that they tend to do away with syntax: "...syntactic structure is merely the characterization of the process of constructing a logical form, rather than a representational level..." — Steedman (2000), p. xi[12]. The main differences are as follows. CCG is a categorial formalism, NLC not. CCG has complex (combinatorial) syntactic types of the form $X\backslash Y$ and X/Y instead of lump types with (what is conventionally viewed as) morphological, syntactic and semantic information. Another difference is CCG's (by default limited) support for word order. (In particular, CCG does not handle concatenations (of terms of types) $\mathrm{c}(X/Z, Y, Z)$ and $\mathrm{c}(Z, Y, X\backslash Z)$, where Y is nonempty[13].) Also, in CCG, NL constructions of sentence level and below can have multiple structures *independently* of (what is traditionally called) interpretational ambiguity.

7.2 Head-Driven Phrase Structure Grammar

It is also useful to compare NLC with a more dedicated syntactic formalism such as Head-Driven Phrase Structure Grammar (HPSG). As a mature formalism that has been implemented for several languages (Pollard and Sag, 1994; DELPH-IN, 2019), HPSG is currently implementation-wise much superior to NLC (which has been implemented only for a fragment of English[14]), so it is appropriate to compare only formalisms. We start with similarities. Both formalisms model parts of semantics, syntax and morphology, but HPSG's scope is much wider, as it covers also

[12] However, as described below, CCG still features syntactic types.

[13] I am not sure whether such concatenations exist in NLs with fixed word order, but a decision to rule them out by default is arbitrary. However, perhaps it would be feasible to introduce a special rule for accommodating Y in this case.

[14] Structurally, the fragment is quite universal. In fact, with a slightly more general notation one can approximate a Universal Grammar (a statement that will make more sense after reading section 9 and recalling that NL semantics is universal).

lexicon and word and morpheme orders. Both formalisms are compositional in that the meaning of a sentence is given by its constituent structure (Carnie, 2012), but the ways of achieving this are very different: HPSG uses attributes (features) and attribute value matrices while NLC uses types and functions. With this the similarities seem to end. HPSG and NLC are fundamentally different kinds of formalisms — the former features an extensive set of attributes, rules and attribute complexes, while the latter has only three rules (introducing atomic, function application and lump types), leaving the specification of types to the implementor. In a sense, there is little to compare, as HPSG is a full-blown *NL grammar formalism* (that has been extended to cover also selectional restrictions) while NLC is a *generic type system for modeling NL compositionality*. Thus, NLC is a much simpler and more general system, and its implementor has significantly more freedom in NL modeling than an implementor of HPSG.

8 Implementing NLC

My experience of implementing NLC is quite limited, as I have so far tried to implement it only in one programming language and have implemented at best a half of NL in terms of its general (or typological) category structure. Below is a test of an implementation of NLC. The test is by type-checking possible NL(C) expressions. The code (from Supplement B) is generously commented and should be self-explanatory.

```
Check PAST throw john. (* "John threw"
type checks -- but not as a sentence: *)
Fail Check PAST throw john: S. (* "At the hut"
can be the 3rd argument of "throw": *)
Check PAST throw john (-s stone)
(at (the hut)). (*..but not the 2nd one:*)
Fail Check PAST throw john (at (the hut)).
(* "In a hut" cannot be
an argument of "throw": *)
Fail Check PAST throw john
(-s stone) (in (a hut)).
(* ..but can be a sentence modifier: *)
Check in (a hut) (PAST throw john (-s stone)).
(* ..and so can "at every hut" and
sentential adverbs like "however": *)
Check at (every hut) (PAST throw john
(-s stone)): S.
Check however (PAST throw john (-s stone)): S.
(* Connectives cannot range over a
sentential and nominal argument: *)
Fail Check and (PAST throw john (a stone))
john. (* ..but can range over nominal: *)
Check and (every john) (all (the (-s boy))).
(* ..or sentential arguments: *)
Check and (PAST walk (-s boy)
(to (all (-s hut)))) (PAST sleep john).
(* ..(also w/ optional arguments omitted): *)
Check and (PAST walk john) (PAST sleep john).

(* Examples of lump types *)
(* "John" is an XP, proper name, male, in
nominative, singular, a physical entity: *)
Check john: XP0 Phy NOM SG (Pn Ml).
(* ..and a limbed entity: *)
Check john: XP0 Lim NOM SG (Pn Ml).
(* "The entire hut and all Johns" is an XP
and physical entity in nominative: *)
Check and (the (entire hut))
(all (-s john)): XP2 Phy NOM _ _. (* ..or
pseudo-accusative (by zero-derivation): *)
Check and (the (entire hut))
(all (-s john)): XP2 Phy ACC' _ _.
(* ..and can be made into a sentient
entity in Lax mode only: *)
Check [and (the (entire hut))
(all (-s john))]: XP2 Sen ACC' _ _.

(* "John threw madly blue stones at the hut
and red limbed boys." has 2 parses: *)
Check madly (PAST throw) john
(blue (-s stone)) (at (and (the hut)
(red [limbed (-s boy)])))): S.
Check PAST throw john
(madly blue (-s stone)) (at (and (the hut)
(red [limbed (-s boy)])))): S.
(* Here we used "[...]" to make a
limbed entity into a physical one. *)

(* We can stack adverbs and adjectives, and
use adjectival and adverbial connectives: *)
Check all ((and madly madly) red (red
[and blue limbed [-s john]])). (* All madly
and madly red, red, blue and limbed Johns *)
```

In Coq, _ is a placeholder for any admissible term or type. A switch in the file the code is taken from allows to choose between Strict and Lax modes, respecting and ignoring selectional restrictions, respectively. The notation [...] interfaces with the current mode. The example omits all technical details like type definitions, etc. These are not instrumental to NLC, as the type system — i.e. one capturing the morphological, syntantic and semantic compositionality of NL with lump types as faithfully as possible — can be implemented in several ways (cf. Supplement A) and different programming languages. The implementation uses only a tiny subset of Coq's features, and its main functionality, theorem proving, is entirely optional here. As I am not at all convinced that Coq is the best language for implementing NLC, I encourage the interested reader to experiment with a programming language of their choice. That being said, statically typed programming languages with a sufficiently complex type system and advanced type inference have some advantages for this kind of work (e.g. in terms of rigor and the similarity of implemented formulas to NL expressions).

9 Implications

The driving force behind NLC has been to correspond to NL as closely as possible. Since ontology (or world knowledge) interfaces with the compositional semantics of NL, it is desirable to formalize some of it in the form of selectional restrictions. We have collapsed syntactic, morphological and semantic compositionality to a single level — to that of the type system. In effect, some types have become syntactic, but the syntax has only two rules: functionality (CTV-Intro) and lumping (LT-Intro). In sum, the paper (and the underlying formalization) have shown that:

(†) A feature of natural language — viz. morphological, syntactic and semantic compositionality — can be reduced to a single level of description.

It is not clear what (†) means, so let us try to explore it further, by (temporarily) assuming that (†) posits a new level of description — call it compositionality — which, moreover, would have to interface with lexical semantics and (what is left of) morphology and syntax. This would be not only theoretically unheard-of (which would be only a mild objection) but would have the undesirable consequence of complicating the general framework of linguistic theory. However, it would have some positive outcomes as well, namely "eliminating" compositional semantics and simplifying morphology and syntax proper. The general theory of natural language would become more complex while three subtheories (morphology, syntax and semantics) would simplify.

Depending on one's outlook on the general theory of natural language, this might seem like a path worth pursuing. However, below I will argue that it is not the only one. The alternative would be to assume that:

(‡) There's nothing "morphological" or "syntactic" about morphological and syntactic compositionality — it is all just semantic compositionality.

Clearly, (†) and (‡) are not mutually exclusive — in fact, (‡) is just a more radical version of (†) (and incidentally, also subsumes (†)). (‡) just conflates the hypothetical new level of description of (†) with compositional semantics. Word and (subword) morpheme order pertain to syntax and morphology, respectively; the compositionality of words, phrases, morphemes and clauses pertains to semantics. As a desirable consequence, we could continue using the existing general theory of natural language with only a few terminological changes. But (how) would (‡) be viable?

A possible justification would make at least two arguments. First, from the theoretical side, morphological, syntactic and semantic compositionality all refer to certain (parts of) knowledge — namely, about morphology, syntax and world, respectively. The only way to have knowledge is by way of meaning, which, given the above, is clearly linguistic. This consideration roots our enterprise in linguistic semantics. Second, from the formalization side, we are using type theory, which is a theory of semantics (broadly defined[15] — cf. (i)). This argument formally corroborates the claim that NLC models only natural language semantics.

Of course, the fact that NL can be modeled this way does not entail that this is the way it works in the brain[16]. So far, our argument has been solely about modeling: It is more parsimonious to model compositionality in a functional type system than e.g. with syntax trees or phrase structure rewrite rules, since the latter cannot, neither separately nor when combined, account for all compositionality. The only advantage of the rewrite rules and syntax trees over the type-theoretical modeling is that they allow, in principle, to capture word order. However, not all syntactic theories support linear order preserving trees (the Chomskian transformational grammar being a case in point — Chomsky (1965, 1981)). Secondly, a word order rule is, differently from compositionality, not a linguistic universal (there are many languages with flexible word order — Dryer (2013)). Incidentally, this also means that *not all natural languages have syntax*.

One thing that seems to emerge from the literature on language processing is the role of syntax as guiding semantic interpretation, or (more figuratively) serving semantics (Kempson et al., 2001; Morrill, 2010; Christiansen and Chater, 2016). Some authors have explicitly argued against syn-

[15] Historically, the semantics of mathematics, more recently also the semantics of programming languages.

[16] Incidentally, there is little sense in trying to make a case of "how language works in the brain", as there is no consensus on this among psycho- and neurolinguists (Chater and Christiansen, 2016).

tax as a separate representational level of linguistic structure (Pulman, 1985; Steedman, 2000). Topographical patterns of brain activation to nouns and verbs are driven not by lexical (grammatical) class but by semantics and word meaning (Moseley and Pulvermüller, 2014). A cognitive architecture with a multi-level (syntactic, semantic, morphological, etc.) NL processing usually requires positing at least as many memory buffers for it (Levinson, 2016), while our short-term memory (obviously recruited in e.g. dialogues) is very limited (Cowan, 2001). These pieces of evidence from language studies also corroborate (‡).

In sum, we hypothesize that combinatorial (im)possibilities in syntax and morphology are better analyzed as belonging to the domain of compositional semantics. Moreover, the conjecture that the traditional boundaries between the levels of description reflect more of a sociological (a division of labor among linguists) than a linguistic fact is not too bold.

If the hypothesis is correct, nothing remains in syntax except word order[17]. Since word order can label (now already semantic) constituents, as in $John_{\text{SUB}}$ $loves$ $Mary_{\text{OBJ}}$, a limited form of syntax-semantics interface is also expected. This accords with the view of the function of syntax as serving semantics, viz. in interpretation disambiguation, which is the primary function of word order constraints (as witnessed in the example above). Syntax-semantics interface is also an appropriate level for phenomena like anaphora and ellipsis. Likewise, morphology proper retains only (subword) morpheme order and fusion, while morphophonological (i.e. morphology-phonology interface) phenomena like e.g. sandhi must also be accounted for. As a result, syntax and morphology emerge as by-products of a contingent conformation to the serial channel over which language is processed.

10 Conclusion

If we interpret proper arguments as nullary functions, all NL expressions up to the sentence level (i.e. morphemes, words, phrases, clauses and sentences) can be interpreted as functions and function applications. The paper presents NLC, a generic functional type system (i.e. one consisting primarily of functions and their applications). NLC is generic in at least two respects: 1. It is applicable to all NLs, and 2. It allows for generic modeling of morphological, syntactic and semantic compositionality. Besides functions, applications and their types, NLC features polymorphic and lump types. The latter are compound types satisfying LT-Intro. Compound types are types which are syntactic compounds of multiple types or their terms (Σ-, Π- and Cartesian product types are examples of compound types). At its core, NLC is a simple system for an integrated modeling of the morphological, syntactic and semantic compositionality of NL with lump types.

The paper also presents an implementation of NLC in Coq (which, unfortunately, is not quite as simple, which may be Coq's fault). The main goal of the implementation was to formalize a reasonably diverse fragment of NL in NLC, with formalized NLC expressions type checking and failing to type check in exactly the same ways that NL expressions pass and fail their acceptability tests. Aside from this goal's feasibility, the implementation shows several things: (1) the viability and simplicity of NLC for modeling NL compositionality, (2) the utility of lump and polymorphic types in NL modeling, and most importantly, (3) the possibility of reducing morphological, syntactic and semantic compositionality to a single level of description. In discussion we have tried to identify this level as semantic compositionality — an interpretation which, besides being supported by results from language processing (Pulman, 1985; Steedman, 2000; Moseley and Pulvermüller, 2014), has interesting implications on NL structure and modeling. In particular, it may reduce syntax and morphology to word and morpheme orders, respectively (with the syntax-semantics and phonology-morphology interfaces reducing correspondingly), with NL architecture taking on a rather different look. This has also implications on linguistic typology, as syntax would, much like morphology before it (Muansuwan, 2002; Grandi and Montermini, 2005; Klamer, 2005), cease to be a logically necessary component of NL.

Acknowledgments

I thank Jason Gross, Hendrik Luuk, Erik Palmgren and Enrico Tassi for their advice. This work has been supported by IUT20-56 and European Regional Development Fund through CEES.

[17] If we do not posit lexicon as a separate level, lexical categories also pertain to morphology or syntax.

References

Nicholas Asher. 2014. Selectional restrictions, types and categories. *Journal of Applied Logic* 12(1):75–87. https://doi.org/10.1016/j.jal.2013.08.002.

Daisuke Bekki and Nicholas Asher. 2013. Logical polysemy and subtyping. In Yoichi Motomura, Alastair Butler, and Daisuke Bekki, editors, *New Frontiers in Artificial Intelligence*, Springer, Berlin, Heidelberg, pages 17–24.

Andrew Carnie. 2012. *Syntax: A Generative Introduction*. Wiley-Blackwell, Malden, MA, 3rd edition. https://www.wiley.com/en-ee/Syntax:+A+Generative+Introduction,+3rd+Edition-p-9780470655313.

Nick Chater and Morten H. Christiansen. 2016. Squeezing through the Now-or-Never bottleneck: reconnecting language processing, acquisition, change, and structure. *Behavioral and Brain Sciences* 39:e91. https://doi.org/10.1017/S0140525X15001235.

Stergios Chatzikyriakidis and Zhaohui Luo. 2014a. Natural language inference in Coq. *Journal of Logic, Language and Information* 23(4):441–480. https://doi.org/10.1007/s10849-014-9208-x.

Stergios Chatzikyriakidis and Zhaohui Luo. 2014b. Natural language reasoning using proof-assistant technology: Rich typing and beyond. In *Proceedings of the EACL 2014 Workshop on Type Theory and Natural Language Semantics (TTNLS)*. Association for Computational Linguistics, Gothenburg, Sweden, pages 37–45. http://www.aclweb.org/anthology/W14-1405.

Stergios Chatzikyriakidis and Zhaohui Luo. 2015. Individuation criteria, dot-types and copredication: A view from modern type theories. In *Proceedings of the 14th Meeting on the Mathematics of Language (MoL 2015)*. Association for Computational Linguistics, pages 39–50. https://doi.org/10.3115/v1/W15-2304.

Stergios Chatzikyriakidis and Zhaohui Luo. 2016. Proof assistants for natural language semantics. In Maxime Amblard, Philippe de Groote, Sylvain Pogodalla, and Christian Retoré, editors, *Logical Aspects of Computational Linguistics. Celebrating 20 Years of LACL (1996–2016)*. Springer, Berlin, Heidelberg, pages 85–98. http://www.cs.rhul.ac.uk/ zhaohui/LACL16PA.pdf.

Noam Chomsky. 1965. *Aspects of the Theory of Syntax*. The MIT Press, Cambridge. http://www.amazon.com/Aspects-Theory-Syntax-Noam-Chomsky/dp/0262530074.

Noam Chomsky. 1981. *Lectures on Government and Binding*. Foris, Dordrecht.

Morten H. Christiansen and Nick Chater. 2016. The Now-or-Never bottleneck: a fundamental constraint on language. *Behavioral and Brain Sciences* 39:e62. https://doi.org/10.1017/S0140525X1500031X.

Robert L. Constable. 2003. Recent results in type theory and their relationship to Automath. In Fairouz D. Kamareddine, editor, *Thirty Five Years of Automating Mathematics*, Springer Netherlands, Dordrecht, pages 37–48.

Robin Cooper. 2005. Records and record types in semantic theory. *Journal of Logic and Computation* 15(2):99–112. https://doi.org/10.1093/logcom/exi004.

Nelson Cowan. 2001. The magical number 4 in short-term memory: a reconsideration of mental storage capacity. *Behavioral and Brain Sciences* 24(1):87–114; discussion 114–85. https://doi.org/10.1017/S0140525X01003922.

DELPH-IN. 2019. The DELPH-IN collaboration. Accessed 18.03.2019. http://www.delph-in.net.

Matthew S. Dryer. 2013. Order of Subject, Object and Verb. In Matthew S. Dryer and Martin Haspelmath, editors, *The World Atlas of Language Structures Online*, Max Planck Institute for Evolutionary Anthropology, Leipzig. http://wals.info/chapter/81.

Nicola Grandi and Fabio Montermini. 2005. Prefix-suffix neutrality in evaluative morphology. In Geert Booij, Emiliano Guevara, Angela Ralli, Salvatore Sgroi, and Sergio Scalise, editors, *On-line Proceedings of the Fourth Mediterranean Morphology Meeting (MMM4), Catania, 21-23 September 2003*. Università degli Studi di Bologna. https://geertbooij.files.wordpress.com/2014/02/mmm4-proceedings.pdf.

N. Gruzitis and D. Dannélls. 2017. A multilingual FrameNet-based grammar and lexicon for controlled natural language. *Lang Resources & Evaluation* 51(1):37–66. https://doi.org/10.1007/s10579-015-9321-8.

Normunds Gruzitis, Peteris Paikens, and Guntis Barzdins. 2012. FrameNet resource grammar library for GF. In Tobias Kuhn and Norbert E. Fuchs, editors, *Controlled Natural Language*, Springer, Berlin, Heidelberg, pages 121–137.

Ruth Kempson, Wilfried Meyer-Viol, and Dov Gabbay. 2001. *Dynamic Syntax: The Flow of Language Understanding*. Blackwell, Oxford.

Marian Klamer. 2005. Explaining structural and semantic asymmetries in morphological typology. In Geert Booij, Emiliano Guevara, Angela Ralli, Salvatore Sgroi, and Sergio Scalise, editors, *On-line Proceedings of the Fourth Mediterranean Morphology Meeting (MMM4), Catania, 21-23 September 2003*. Università degli Studi di Bologna. https://geertbooij.files.wordpress.com/2014/02/mmm4-proceedings.pdf.

Joachim Lambek. 1958. The mathematics of sentence structure. *The American Mathematical Monthly* 65(3):154–170.

Stephen C. Levinson. 2016. "Process and perish" or multiple buffers with push-down stacks? *Behavioral and Brain Sciences* 39:e81. https://doi.org/10.1017/S0140525X15000862.

Zhaohui Luo. 2010. Type-theoretical semantics with coercive subtyping. In *Semantics and Linguistic Theory*. Vancouver, volume 20, pages 38–56.

Zhaohui Luo. 2011. Contextual analysis of word meanings in type-theoretical semantics. In Sylvain Pogodalla and Jean-Philippe Prost, editors, *Logical Aspects of Computational Linguistics*. Springer Berlin Heidelberg, Berlin, Heidelberg, pages 159–174.

Zhaohui Luo. 2014. Formal semantics in modern type theories: is it model-theoretic, proof-theoretic, or both? In Nicholas Asher and Sergei Soloviev, editors, *Logical Aspects of Computational Linguistics 2014 (LACL 2014)*, Springer, Berlin, Heidelberg, number 8535 in LNCS, pages 177–188.

Erkki Luuk. 2010. Nouns, verbs and flexibles: implications for typologies of word classes. *Language Sciences* 32(3):349–365. https://doi.org/10.1016/j.langsci.2009.02.001.

Richard Montague. 2002. The proper treatment of quantification in ordinary English. In Paul Portner and Barbara H. Partee, editors, *Formal Semantics: The Essential Readings*, Blackwell, Oxford, pages 17–34.

Glyn Morrill. 2010. *Categorial grammar: Logical syntax, semantics, and processing*. Oxford University Press, Oxford.

Rachel L. Moseley and Friedemann Pulvermüller. 2014. Nouns, verbs, objects, actions, and abstractions: Local fMRI activity indexes semantics, not lexical categories. *Brain and Language* 132:28 – 42. https://doi.org/10.1016/j.bandl.2014.03.001.

Nuttanart Muansuwan. 2002. *Verb Complexes in Thai*. Ph.D. thesis, University at Buffalo, The State University of New York. https://arts-sciences.buffalo.edu/content/dam/arts-sciences/linguistics/AlumniDissertations/Muansuwan dissertation.pdf.

Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.

Stephen G. Pulman. 1985. A parser that doesn't. In *Proceedings of the 2nd European Meeting of the Association for Computational Linguistics, Geneva: ACL*. pages 128–135. https://www.aclweb.org/anthology/E85-1019.

Aarne Ranta. 1994. *Type-theoretical grammar*. Clarendon Press, Oxford; New York.

Aarne Ranta. 2004. Grammatical Framework: a type-theoretical grammar formalism. *The Journal of Functional Programming* 14(2):145–189. https://doi.org/10.1017/S0956796803004738.

Mark Steedman. 2000. *The Syntactic Process*. MIT Press, Cambridge, MA, USA.