Lambek Grammars as Second-order Abstract Categorial Grammars

Oleg Kiselyov[0000−0002−2570−2186] and Hoshino Yuya

Tohoku University, Japan
oleg@okmij.org

Abstract. We demonstrate that for all practical purposes, Lambek Grammars (LG) are *strongly* equivalent to Context-Free Grammars (CFG) and hence to second-order Abstract Categorial Grammars (ACG). To be precise, for any Lambek Grammar LG there exists a second-order ACG with a second-order lexicon such that: the set of LG derivations (with a bound on the 'nesting' of introduction rules) is the abstract language of the ACG, and the set of yields of those derivations is its object language. Furthermore, the LG lexicon is represented in the abstract ACG signature with no duplications. The fixed, and small, bound on the nesting of introduction rules seems adequate for natural languages. One may therefore say that ACGs are not merely just as expressive as LG, but strongly equivalent.
The key is the algebraic description of Lambek Grammar derivations, and the *avoidance* of the Curry-Howard correspondence with lambda calculus.

Keywords: Lamkek Grammar · Context-Free Grammar · Pentus Construction · ACG.

1 Introduction

Expressing a Lambek Grammar (LG) as an Abstract Categorial Grammar (ACG) is a sort of a problem that on the surface is either impossible or trivial, with unfolding subtleties and depth – the problem that just does not go away. Lambek Grammar with its directional types is based on logic with directional implications without any exchange rule. In contrast, ACG uses ordinary arrow types, and its underlying implicative fragment of multiplicative linear logic is commutative. No matter what tricks one may play, the fundamental distinction inevitably comes to haunt us, as Kubota and Levin [8] and Moot [9] have claimed: "The best approximations that we can obtain all suffer from overgeneration because non-commutativity is insufficiently enforced." [9, §7.2]. In fact, analyzing right-node raising in an ACG formalism without directional

types while avoiding overgeneration has been posed as a challenge to the first author by Yusuke Kubota at ESSLLI 2013.

On the other hand, the problem seems trivial: as Pentus showed [10], any LG is weakly equivalent to a context-free grammar (CFG), and CFGs are trivially representable as ACGs [3]. Weak equivalence means that the two grammars generate the same set of strings. The correspondence of derivations is a different, and subtle matter, investigated by Kanazawa, Salvati [6, 4], De Groote [1] and others [12]. The latest result is De Groote's construction of an ACG that reproduces both derivations and the yields of an LG [1]. However, that ACG is third order, and still suffers from redundancies that arise in Pentus-like constructions. Furthermore, the paper [1] notes that the strong equivalence of ACG and LG cannot be guaranteed in every case.

Our contribution is the general strong equivalence of LG and ACG:

Theorem 1. *For any LG and the natural number n, there exists a second-order ACG with a second-order lexicon whose abstract language is all and only LG derivations of the distinguished type of hyp-rank n (to be defined below) and whose object language is the yields of those LG derivations. The LG lexicon enters the ACG signature with no duplications, let alone exponential explosions.*

As a corollary, we answer the Kubota challenge. As another corollary,

Corollary 1. *For any LG and the natural number n, there exists a context-free grammar (CFG) whose parses are all and only start-type LG derivations of hyp-rank n. (In fact, CFG parse trees are LG derivation trees, written 'upside down'.)*

The paper presents the construction of the CFG and ACG from an LG and argues, in §5, that the hyp-rank qualification is irrelevant in practice. The key is the algebraic approach to LG derivations and ACG and avoiding the Curry-Howard correspondence. We do *not* regard directional types as function (arrow) types.

The structure of the paper is as follows. The next section reminds Lambek calculus and grammars and introduces their different but provably equivalent presentation, which is easier to characterize algebraically. §3 gives an unconventional, algebraic presentation of second-order abstract categorial grammars. The algebraic presentation immediately relates ACG with LG derivations, leading to the main result of the paper. The strong equivalence of ACG and LG means the absence of overgeneration. §3.1 explicates the reason preventing the overgeneration, and, on

the concrete examples from Moot [9] and the Kubota challenge, demonstrates the descriptive adequacy of second-order ACGs. In §4 we examine related work, in particular, [1]. We also show how our presentation of LG avoids the most difficult issues in the Pentus construction [10]. §5 discusses hyp-rank and the algebraic presentation of LG.

2 Lambek Grammars, Derivations, and Algebras

First we recall the needed definitions of Lambek calculus and grammar, define the hyp-rank and introduce the running example. §2.1 later presents the variation, the calculus LA, and its algebraic characterization.

A grammar is a description of a language, that is, of a set of finite strings built from a finite fixed set of 'words' (so-called alphabet). We write ϵ for the empty string and $+$ for string concatenation. In Lambek Grammars (LG) – and type logical grammars that followed – the string building rules are expressed through a deductive system. In the case of LG, the deductive system is the (associative) Lambek calculus L presented below, in the Gentzen-style natural deduction form (from [11, §2.2.2]).

$$\begin{array}{lll} \text{Primitive types } P & ::= & s, n, np \\ \text{Types} & A, B, C & ::= P \mid A\backslash B \mid B/A \\ \text{Environments} & \Gamma, \Delta & ::= A \mid A, \Gamma \mid \Gamma, A \\ \text{Judgements} & \Gamma \vdash A & \end{array}$$

$$\frac{\Delta \vdash B/A \quad \Gamma \vdash A}{\Delta, \Gamma \vdash B}\,/e \qquad \frac{\Gamma, A \vdash B}{\Gamma \vdash B/A}\,/i$$

$$\frac{\Gamma \vdash A \quad \Delta \vdash A\backslash B}{\Gamma, \Delta \vdash B}\,\backslash e \qquad \frac{A, \Gamma \vdash B}{\Gamma \vdash A\backslash B}\,\backslash i$$

$$\frac{}{A \vdash A}\,Var$$

The formulas of L are called syntactic types, for which we use the metavariables A, B, and C. They are inductively built from the primitive types s, n, and np using the left- and right- slashes. As the convenient abbreviation, vp stands for $np\backslash s$, tv for vp/np, det for np/n, rel for $(n\backslash n)/(s/np)$ and pp for $n\backslash n$. The metavariables Γ and Δ stand for an environment: a non-empty sequence of types. Furthermore, $A, (B, C)$ and $(A, B), C$ represent the same environment; an environment is hence just a linear sequence of types. Besides associativity, there are no other (structural) rules about the environments; in particular, there is no exchange

rule: the order of types in an environment is significant. We define a partial order on types $A \prec B$ (pronounced: 'A is a subformula of B') as $A \prec A$, and $A \prec B/C$, $A \prec B\backslash C$ whenever $A \prec B$ or $A \prec C$. We say A is a subformula of a collection of types if it is a subformula of some type in the collection. As an example of L, Fig.1 shows the derivation of the judgement $det, n, rel, np, tv, vp \vdash s$.

$$\dfrac{\dfrac{det \vdash det \qquad \dfrac{n \vdash n \qquad \dfrac{rel \vdash rel \qquad \dfrac{\dfrac{\dfrac{np \vdash np \qquad \dfrac{tv \vdash tv \qquad np \vdash np}{tv, np \vdash vp}\backslash_e}{np, tv, np \vdash s}/_i}{np, tv \vdash (s/np)}/_e}{rel, np, tv \vdash pp}\backslash_e}{n, rel, np, tv \vdash n}/_e}{det, n, rel, np, tv \vdash np} \qquad vp \vdash vp}{det, n, rel, np, tv, vp \vdash s}\backslash_e$$

Fig. 1. Gentzen-style, natural deduction derivation in L

A grammar based on L – Lambek Grammar (LG) – is a tuple $(\mathcal{L}_L, A_s)$ of the lexicon and the *initial type* (which is often s). The lexicon $\mathcal{L}_L$ defines the alphabet of the grammar and assigns to each word of the alphabet the corresponding L type. Fig.2 shows the sample lexicon, also to be denoted by $\mathcal{L}_L$.

John : np book : n the : det that : rel read : tv vanished : vp

Fig. 2. The LG lexicon $\mathcal{L}_L$ for the running example

A non-empty string $w_1 \dots w_n$ (where w_i is a word of the alphabet) is in the language of the grammar $(\mathcal{L}_L, A_s)$ just in case the judgement $A_1, \dots, A_n \vdash A_s$ is derivable in L, where A_i is the type assigned by $\mathcal{L}_L$ to w_i. The derivation in Fig.1 thus shows that the language of the grammar ($\mathcal{L}_L$,s) includes the string "The book that John read vanished," which will be our running example.

The derivation in Fig.1 contains a number of hypotheses (on the left-side of the turnstile), only one of which is discharged, by the introduction rule. The maximum number of to-be-discharged hypotheses in any judgement of a derivation is called the *hyp-rank* of the derivation. (The derivation in Fig.1 hence has the hyp-rank one).

2.1 The Calculus LA

To conveniently view Lambek derivations as an algebra, and to later relate them to ACG, we give a different but provably equivalent presentation of L and LG, to be called LA. The only differences from L are the addition of marks $\centerdot$ to the environment and a different treatment of the lexicon. The new syntax of the environment is:

$$\Gamma, \Delta ::= \centerdot \mid A \mid A, \Gamma \mid \Gamma, A$$

We write $\bullet$ for a sequence of one or more consecutive marks $\centerdot$, and $\breve{\Gamma}$, $\breve{\Delta}$ for a possibly empty mark-free sequence of types.

The grammar based on the calculus (also to be called LA) adjoins to LA a set of axioms – the LA lexicon $\mathcal{L}_A$ – and designates one of the types as initial (usually, s). $\mathcal{L}_A$ is a different presentation of the LG lexicon: the mapping of a word to a type is written as an axiom, whose name is the word and whose conclusion is the corresponding type. If w is a word in $\mathcal{L}_A$, we write $\mathcal{L}_A(w)$ for its type; the notation extends to sequences of words. Shown below are three (for the sake of space) axioms of $\mathcal{L}_A$ that correspond to $\mathcal{L}_L$ of the running example:

$$\frac{}{\centerdot \vdash np}\ john \qquad \frac{}{\centerdot \vdash n}\ book \qquad \frac{}{\centerdot \vdash tv}\ read$$

The mark thus 'marks the place' of a lexical entry axiom in the derivation. Figure 3 gives the LA derivation for the running example.

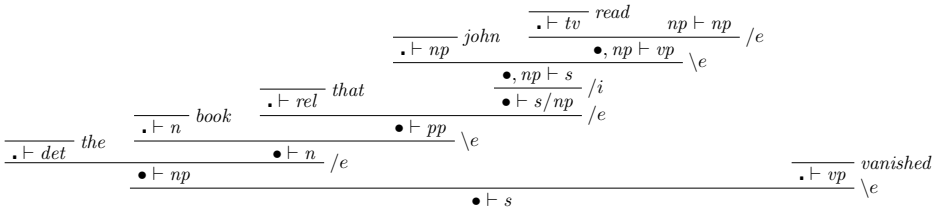

Fig. 3. Sample derivation in LA

If t is an LA derivation (tree), its *fringe* $\mathcal{F}(t)$ is a string of words naming the axioms that appear in its derivation. It is inductively defined as follows:

- if t is an axiom named w, $\mathcal{F}(t)$ is w;
- if t is a Var axiom $A \vdash A$, $\mathcal{F}(t)$ is ϵ

- if the last rule in t is $\backslash i$ or $/i$ with the premise t', then $\mathcal{F}(t)$ is $\mathcal{F}(t')$;
- if the last rule in t is $\backslash e$ or $/e$ with the premises t_1 and t_2, then $\mathcal{F}(t)$ is $\mathcal{F}(t_1)+\mathcal{F}(t_2)$.

A non-empty string $w_1 \dots w_n$ is said to belong to the language of an LA grammar with the lexicon $\mathcal{L}_A$ and the initial type A_s just in case there is an LA derivation of $\bullet \vdash A_s$ whose fringe is $w_1 \dots w_n$.

The notion of *normal derivations* in LA is the same as in L [11, §2.8], as should be clear from the strong equivalence shown below. That is, an LA derivation is called normal if it does not contain an introduction rule for the type B/A or $A\backslash B$, immediately followed by the rule that eliminates A. As in L, LA derivations can always be normalized [11, §2.8]. Therefore, we restrict our attention to normal derivations only. Also like in L, normal LA derivations enjoy the subformula property: any type that appears within a normal derivation $\Gamma \vdash A_s$ is a subformula of Γ, A_s or the set of lexicon types.

2.2 Strong Equivalence of LG and LA Grammars

Proposition 1. *If t is an LA derivation of $\breve{\Gamma}, \bullet, \breve{\Delta} \vdash A$ (given the lexicon $\mathcal{L}_A$), then there exists the L derivation $\breve{\Gamma}, \mathcal{L}_A(\mathcal{F}(t)), \breve{\Delta} \vdash A$.*

The proof is an easy induction on the structure of t. Indeed,

- If t is an axiom $\centerdot \vdash A$ named w, the corresponding L judgement is $A \vdash A$, which is the Var axiom of L.
- If the last rule of t is $/i$ with the conclusion $\breve{\Gamma}, \bullet, \breve{\Delta} \vdash B/A$, its premise must be $\breve{\Gamma}, \bullet, \breve{\Delta}, A \vdash B$ (whose derivation is to be called t'). By the inductive hypothesis, there exists an L derivation $\breve{\Gamma}, \mathcal{L}_A(\mathcal{F}(t')), \breve{\Delta}, A \vdash B$, which can then be extended with the $/i$ rule to $\breve{\Gamma}, \mathcal{L}_A(\mathcal{F}(t)), \breve{\Delta} \vdash B/A$ (keeping in mind that $\mathcal{F}(t) = \mathcal{F}(t')$).
- Suppose the last rule of t is $/e$ with the conclusion $\breve{\Gamma}, \bullet, \breve{\Delta} \vdash B$ and the premises t_1 and t_2. There are three cases to consider. In the first, t_1 has the form $\breve{\Gamma}, \bullet \vdash B/A$ and t_2 has the form $\bullet, \breve{\Delta} \vdash A$. Applying the inductive hypothesis to both and then $/e$ gives $\breve{\Gamma}, \mathcal{L}_A(\mathcal{F}(t)), \breve{\Delta} \vdash B$, keeping in mind that $\mathcal{F}(t) = \mathcal{F}(t_1)+\mathcal{F}(t_2)$. In the second case, t_1 has the form $\Gamma_1 \vdash B/A$ and t_2 has the form $\breve{\Gamma}_2, \bullet, \breve{\Delta} \vdash A$ where Γ_1 is unmarked and non-empty. Then t_1 is the ordinary L derivation, using no axioms of $\mathcal{L}_A$, with no marks in its environment and with the empty fringe. We reach the conclusion by applying the induction hypothesis to t_2 only, and then the $/e$ rule to get the final L derivation. The remaining case is symmetrical.

– The cases of $\backslash e$ and $\backslash i$ as the last rules of t are symmetrical.

Proposition 2. *If $\Gamma \vdash A$ is derivable in L, then there exists a lexicon $\mathcal{L}_A$ and an LA derivation t of $\breve{\Gamma}, \bullet, \breve{\Delta} \vdash A$ such that $\breve{\Gamma}, \mathcal{L}_A(\mathcal{F}(t)), \breve{\Delta}$ is Γ.*

Consider an L derivation of $\Gamma \vdash A$ where Γ, which must be non-empty, is $A_1, \ldots, A_n$. Each type A_i must come from a Var axiom $A_i \vdash A_i$ that is used at some point in the derivation. Pick from Γ a non-empty consecutive sequence of types $A_k, A_{k+1}, \ldots, A_l$ with $l > k$, and build the lexicon $\mathcal{L}_A$ with the axioms $\blacksquare \vdash A_i$ named by distinct words w_i, where $i = k..l$. For each such A_i we replace the Var axiom that introduced that A_i into Γ with the corresponding axiom from $\mathcal{L}_A$ – replacing the corresponding occurrences of A_i with $\blacksquare$. It is easy to see that the result is the valid LA derivation with the fringe whose types are $A_k, A_{k+1}, \ldots, A_l$.

The easy corollary from the two propositions is that LG and LA describe the same set of strings: they are weakly equivalent. The examination of the proofs lets us conclude the stronger result: LG and LA derivations for the same string have the same shape, and differ only in the 'kind' of some of their axioms, Var vs. lexicon. Thus LG and LA are *strongly equivalent*. We may thus speak of LA derivations as Lambek grammar derivations.

2.3 The Algebra of LA Derivations

One may regard the derivation trees like those in Fig.3 as a multi-sorted algebra, to be called $\mathcal{AL}_{D_1}$. Its carrier sets are LA derivations; Fig.4 shows the signature Σ_{AL_1} of the operations. (Notably, it includes the lexicon $\mathcal{L}_L$, without any duplications: see the first column of Fig.4.) There, $\langle \bullet, np; vp \rangle$, etc. is the notation for *atomic* sorts (that is, types); although angular brackets, commas, semicolons might suggest an internal structure, it is only for the convenience of the reader. In the formalism, the whole complicated symbol denotes an atomic type (sort) without any separately interpreted components.

The carriers of $\mathcal{AL}_{D_1}$ are LA derivations with no more than one dischargeable np hypothesis at a time. The construction of $\mathcal{AL}_{D_n}$ for derivations of any other fixed hyp-rank is analogous. For example, the general $\mathcal{AL}_{D_2}$ adds to $\mathcal{AL}_{D_1}$ the operations such as $\langle np, \bullet; tv \rangle \to \langle \bullet, np; np \rangle \to \langle np, \bullet, np; vp \rangle$ and $\langle \bullet; tv \rangle \to \langle \bullet, np, np; np \rangle \to \langle \bullet, np, np; vp \rangle$. There are many such operations, but their number is finite (see below). We can build an algebra whose carriers are all LA derivations with no restrictions; it will have an infinite number of operations: all instances of a

john	$: \langle \bullet; np \rangle$	evp	$: \langle \bullet; np \rangle \to \langle \bullet; vp \rangle \to \langle \bullet; s \rangle$
book	$: \langle \bullet; n \rangle$	enn	$: \langle \bullet; n \rangle \to \langle \bullet; pp \rangle \to \langle \bullet; n \rangle$
the	$: \langle \bullet; det \rangle$	edp	$: \langle \bullet; det \rangle \to \langle \bullet; n \rangle \to \langle \bullet; np \rangle$
that	$: \langle \bullet; rel \rangle$	etv	$: \langle \bullet; tv \rangle \to \langle \bullet; np \rangle \to \langle \bullet; vp \rangle$
read	$: \langle \bullet; tv \rangle$	ehtv	$: \langle \bullet; tv \rangle \to \langle np; np \rangle \to \langle \bullet, np; vp \rangle$
vanished	$: \langle \bullet; vp \rangle$	hnp	$: \langle np; np \rangle$
		ehvp	$: \langle \bullet; np \rangle \to \langle \bullet, np; vp \rangle \to \langle \bullet, np; s \rangle$
		irnp	$: \langle \bullet, np; s \rangle \to \langle \bullet; s/np \rangle$
		erel	$: \langle \bullet; rel \rangle \to \langle \bullet; s/np \rangle \to \langle \bullet; pp \rangle$

Fig. 4. The signature Σ_{AL_1}

finite set of schematic operations (which are the restatement of the LA inference rules).

Let $\mathbb{L}_n$ be the language of an LA grammar whose derivations have the hyp-rank n. The hyp-rank is the bound on the nesting of the introduction rules – or, the bound on the length of Γ of any judgement used in a derivation. The hyp-rank does not limit the total number of the introduction (or, elimination, for that matter) rules that may occur in a derivation. Therefore, $\mathbb{L}_n$ is generally infinite. Nevertheless, all its derivations can be performed with a finitely many instances of inference rules. Indeed, if t is a normal derivation $\bullet \vdash A_s$, all types of all judgements within t are subformulas of A_s or the types of the lexicon. Therefore, the set of distinct types appearing within all (potentially infinite many) derivations for $\mathbb{L}_n$ is finite. Furthermore, each judgement within a derivation has no more that n hypotheses. Therefore, the total number of distinct judgements, and hence the distinct instances of inference rules, within all derivations of $\mathbb{L}_n$ is also finite.

It is easy to see $\mathcal{AL}_{D_1}$ is an initial algebra of the signature Σ_{AL_1}, and hence represents all and only the LA (and correspondingly, LG) derivations of the given lexicon (with a single dischargeable np hypothesis).

One may view Σ_{AL_1} as a CFG. For example, the type of the operation 'evp' may be viewed as the production $\langle \bullet; s \rangle \to \langle \bullet; np \rangle \ \langle \bullet; vp \rangle$. The grammar is almost in Chomsky Normal Form (it would be in CNF if we substitute-out the productions that correspond to the unary rule 'irnp').

3 ACG, Algebraically

We now define an algebraic ACG: a subset of second-order ACGs [2, 5], and relate it with LA. An algebraic ACG $\mathcal{G}$ is a quadruple of two algebraic signatures Σ_A (called 'abstract') and Σ_O (called 'object'), a

morphism $\mathcal{L}$ (called lexicon) and a sort s of the abstract signature called 'the distinguished type'. $\mathcal{L}$ is a morphism from an initial algebra of Σ_A to the initial algebra of Σ_O that commutes with Σ_A's operations. Let $\mathcal{I}(\mathcal{G})$ be the word algebra of Σ_A. Its carrier of the sort s is called the abstract language generated by $\mathcal{G}$. The object language of $\mathcal{G}$ is the image of the abstract language by $\mathcal{L}$.

In the original De Groote's definition of ACG [2], the abstract language is taken to be the set of all closed linear lambda terms of the type s built over a (generally higher-order) signature. In the simplest second-order ACG, the signature is algebraic. Furthermore, a *normal* lambda term of the primitive type s over such signature has no lambda-abstractions. Thus our algebraic ACG fits with the original second-order ACG definition.

In the following, let $\mathcal{G}$ be a particular algebraic ACG whose abstract signature is Σ_{AL_1}, the object signature is the string signature defined below, and the distinguished type is s. The zero-arity operations of Σ_{AL_1} are precisely the LG lexicon, Fig.2; the other operations are determined by the hyp-rank and the set of lexicon categories – both of which are small, in natural languages. The abstract language of $\mathcal{G}$ is, by definition, the set of terms, such as

```
evp(edp(the, enn(book,
  erel(that, irnp(ehvp(john, ehtv(read,hnp))))))), vanished)
```

which is an encoding of the LG derivation in Fig.1. Since $\mathcal{I}(\mathcal{G})$ is also an initial algebra and hence isomorphic to $\mathcal{AL}_{D_1}$, the abstract language of $\mathcal{G}$ represents all and only derivations of LG (with the hyp-rank restriction).

The string signature has only one sort: string. Its constants are john and book, etc., for each lexical item (Fig.2), plus ϵ and $+$. The morphism from $\mathcal{I}(\mathcal{G})$ is defined in Fig.5. (In conventional ACG terms, the lexicon can be called second-order.) In particular, $\mathcal{L}$ maps the abstract language term above to (the+(book+(that+(ϵ+(john+(read+ϵ))))))+vanished. It is easy to see the morphism computes the fringe of the LA derivation tree: in other words, it computes the yield of the corresponding LG derivation.

$\mathcal{L}(\texttt{john}) \mapsto \mathsf{john}$ $\quad \mathcal{L}(\texttt{hnp}) \mapsto \epsilon$ $\quad \mathcal{L}(\texttt{irnp}(\mathrm{x})) \mapsto \epsilon + \mathcal{L}(\mathrm{x})$
$\mathcal{L}(\texttt{evp}(\mathrm{x},\mathrm{y})) \mapsto \mathcal{L}(\mathrm{x}) + \mathcal{L}(\mathrm{y})$

Fig. 5. The lexicon $\mathcal{L}$: the mapping from $\mathcal{I}(\mathcal{G})$ to the string language. Only the representative mappings are shown. The others are analogous.

3.1 The Absence of Overgeneration

We have just demonstrated that an algebraic ACG describes the same language as the corresponding LG of a fixed hyp-rank and hence does not overgenerate. Since overgeneration is a serious problem in (naive) ACGs [9], let us discuss how it could arise and how it is prevented in algebraic ACGs. We use concrete examples from [9] and the Kubota challenge.

We start with one of the examples from [9, §7.2]: sentences with adverbs. The base sentence is "John hit Mary", which is in the language of $\mathcal{G}$ (after adding to Σ_{AL_1} the lexical entries mary: $\langle \bullet; np \rangle$ and hit: $\langle \bullet; tv \rangle$). The corresponding abstract term is `evp(john, etv(hit,mary))`. After adding "deliberately" with the syntactic type vp/vp, "John deliberately hit Mary" should become recognizable. It does indeed, after the following additions to Σ_{AL_1}:

$$\text{deliberately} : \langle \bullet; vp/vp \rangle \qquad \text{evpvp} : \langle \bullet; vp/vp \rangle \to \langle \bullet; vp \rangle \to \langle \bullet; vp \rangle$$

of which only the first is the lexical entry. The LG derivation is represented by the abstract term `evp(john, evpvp(deliberately,etv(hit,mary)))`.

The challenge itself is avoiding overgeneration. Moot shows in [9] that a naive ACG generates not just the above sentence but also "John deliberately Mary hit" and "Mary John hit deliberately". Let us see how the former could come about. The signature Σ_{AL_1} has no combinators to combine 'mary' with 'hit'. We can introduce a np hypothesis and build `irnp(ehvp(mary,ehtv(hit,hnp)))` of the sort $\langle \bullet; s/np \rangle$ – similar to the analysis of "john read" in our running example. In the naive ACG this term has the type indistinguishable from vp (which is $np\backslash s$), and hence can be combined with 'deliberately' and then with 'john', leading to the overgenerated sentence. Such derivation is, however, impossible in $\mathcal{G}$: the sort $\langle \bullet; s/np \rangle$ is different from $\langle \bullet; vp \rangle$; the combination with 'deliberately' is not possible. The second overgeneration example by Moot is also not derivable in $\mathcal{G}$.

Thus although ACGs do not have directional types, the overgeneration can still be prevented, if directional syntactic types such as $np\backslash s$ are mapped to atomic ACG types (viz., $\langle \bullet; np\backslash s \rangle$) rather than function types. The key, hence, is avoiding the Curry-Howard correspondence, prominently present in other approaches [1, 4]. Using rich set of atomic types does *not* imply duplicating lexical entries: Σ_{AL_1} uses just as many lexical entries as the corresponding LG lexicon.

The Kubota challenge is analyzing right-node-raising such as "John loves and Bill hates Mary" without overgeneration. The original sentence

is recognized by $\mathcal{G}$ after adding (besides the obvious lexical entries for 'bill', 'loves' and 'hates'):

$\text{and} : \langle \bullet; (rn\backslash rn)/rn \rangle$ $\quad$ $\text{eandr} : \langle \bullet; (rn\backslash rn)/rn \rangle \to \langle \bullet; rn \rangle \to \langle \bullet; rn\backslash rn \rangle$
$\text{eandl} : \langle \bullet; rn \rangle \to \langle \bullet; rn\backslash rn \rangle \to \langle \bullet; rn \rangle$ $\quad$ $\text{esnp} : \langle \bullet; rn \rangle \to \langle \bullet; np \rangle \to \langle \bullet; s \rangle$

(we write rn for s/np.) Here is the derivation

```
esnp(eandl(irnp(ehvp(john, ehtv(loves,hnp))),
     eandr(and, irnp(ehvp(bill, ehtv(hates,hnp))))), mary)
```

Crucially, "*Mary John loves and Bill hates" is not recognizable: $\langle \bullet; s/np \rangle$ is different from $\langle \bullet; vp \rangle$ as explained earlier, and the only combinator that accepts the arguments of the types $\langle \bullet; s/np \rangle$ and $\langle \bullet; np \rangle$ takes them in the shown order.

4 Related Work

The most closely related is the work by De Groote [1]. His approach is based on a Pentus-like construction connecting LG and CFG, Kanazawa and Salvati's characterization of that construction [6], a novel method of interpreting LG lexicon as linear lambda terms, and lexicalization of second-order ACGs [13, 7].

To clearly see the differences between De Groote's and our approaches, let us take the running example of [1]: "Every man who loves some woman loves every woman." It is recognized by LG in a hyp-rank–zero derivation with the following lexicon

$\text{man} : n$ $\quad$ $\text{woman} : n$ $\quad$ $\text{some} : det$ $\quad$ $\text{every} : det$ $\quad$ $\text{loves} : tv$ $\quad$ $\text{who} : pp/vp$

(The lexicon shown in [1, Fig.1] has an extra entry for 'whom' that is not used in the running example.)

The most insightful is the comparison of our algebraic ACG with the intermediate result of De Groote's derivation, which he dubs $LDER$: see Fig.6. The differences show already in the abstract signature: $LDER$ is larger, reflecting the fact that Pentus-like constructions produce (highly) redundant grammars. (Even though $LDER$ relied on a particular case of the Pentus construction with less redundancy than the general case). The most significant differences are in the lexicon. The $LDER$ lexicon is clearly third order, producing lambda-terms. Our lexicon is second order, and outputs strings (the yield of the grammar). The final result [1, Figs. 12 and 13] also has the second-order string lexicon, but a *third-order* ACG.

One of the most difficult parts of the Pentus proof [10] is demonstrating that an L derivation $A_1, \ldots, A_n \vdash A_{n+1}$ for an arbitrary n

Abstract signature, [1]

$\text{prod}_0 : \texttt{<}det\texttt{>} \to \texttt{<}n\texttt{>} \to \texttt{<}np\texttt{>}$
$\text{prod}_1 : \texttt{<}tv\texttt{>} \to \texttt{<}np\texttt{>} \to \texttt{<}np\texttt{>} \to \texttt{<}s\texttt{>}$
$\text{prod}_2 : \texttt{<}pp/vp\texttt{>} \to \texttt{<}vp\texttt{>} \to \texttt{<}n\texttt{>} \to \texttt{<}n\texttt{>}$
$\text{prod}_4 : \texttt{<}tv\texttt{>} \to \texttt{<}np\texttt{>} \to \texttt{<}vp\texttt{>}$
$\text{prod}_7 : \texttt{<}det\texttt{>} \to \texttt{<}n/np\texttt{>} \to \texttt{<}np/np\texttt{>}$
$\text{prod}_8 : \texttt{<}pp/vp\texttt{>} \to \texttt{<}tv\texttt{>} \to \texttt{<}n\texttt{>} \to \texttt{<}n/np\texttt{>}$
$\text{prod}_9 : \texttt{<}tv\texttt{>} \to \texttt{<}np/np\texttt{>} \to \texttt{<}tv\texttt{>}$
$\text{man} : \texttt{<}n\texttt{>}$
other lexical entries elided

Abstract signature, this paper

$\text{edp} : \langle \bullet; det\rangle \to \langle \bullet; n\rangle \to \langle \bullet; np\rangle$
$\text{evp} : \langle \bullet; np\rangle \to \langle \bullet; vp\rangle \to \langle \bullet; s\rangle$
$\text{etv} : \langle \bullet; tv\rangle \to \langle \bullet; np\rangle \to \langle \bullet; vp\rangle$
$\text{enn} : \langle \bullet; n\rangle \to \langle \bullet; pp\rangle \to \langle \bullet; n\rangle$
$\text{esrel} : \langle \bullet; pp/vp\rangle \to \langle \bullet; vp\rangle \to \langle \bullet; pp\rangle$

$\text{man} : \langle \bullet; n\rangle$
other lexical entries elided

Object signature, [1]

$\mathsf{man},\mathsf{woman} : n$
$\mathsf{some},\mathsf{every} : n \to np$
$\mathsf{loves} : np \to np \to np$
$\mathsf{who} : (np \to s) \to n \to n$

Object signature, this paper

$\mathsf{man},\mathsf{woman},\mathsf{some},\mathsf{every},\mathsf{loves},\mathsf{who} : string$
$+ : string \to string \to string$

Lexicon, [1]

$\text{prod}_0 := \lambda xy.\, xy$
$\text{prod}_1 := \lambda xyz.\, xyz$
$\text{prod}_2 := \lambda wxy.\, w(\lambda z.\, xz)y$
$\text{prod}_8 := \lambda vwxy.\, v(\lambda z.\, wyz)x$
$\ldots$
$\text{man} := \mathsf{man}$
$\ldots$

Lexicon, this paper

$\mathcal{L}(\text{edp(x,y)}) \mapsto \mathcal{L}(\text{x}) + \mathcal{L}(\text{y})$
$\mathcal{L}(\text{evp(x,y)}) \mapsto \mathcal{L}(\text{x}) + \mathcal{L}(\text{y})$
$\mathcal{L}(\text{etv(x,y)}) \mapsto \mathcal{L}(\text{x}) + \mathcal{L}(\text{y})$
$\mathcal{L}(\text{enn(x,y)}) \mapsto \mathcal{L}(\text{x}) + \mathcal{L}(\text{y})$
$\mathcal{L}(\text{esrel(x,y)}) \mapsto \mathcal{L}(\text{x}) + \mathcal{L}(\text{y})$
$\mathcal{L}(\text{man}) \mapsto \mathsf{man}$
$\ldots$

Fig. 6. Comparison of the ACG grammar *LDER* of [1] and the algebraic ACG of the present paper, for the running example of [1]. The *LDER* grammar is cited from Figs.4-7 of [1], after removing the unused entry for 'whom' and adjusting the notation.

can be constructed, using only the cut rule, from the the derivations of $A_1, \ldots, A_m \vdash A_{m+1}$ where m is only 1 or 2. Limiting the number of hypotheses in all judgements of L to, say, two, limits the length of strings in the LG language also to two. In LA, however, the number of marks $\bullet$ (collapsed into $\bullet$) does not count for the purpose of hyp-rank. Therefore, we may impose the hyp-rank 2 and still generate an infinite set of strings. In fact, as we argue below, the hyp-rank of two or three may be sufficient as far as natural languages are concerned. Therefore, LA side-steps the main difficulty of the Pentus construction.

Moot's [9] is the comprehensive study of type logical grammars, naive ACG and lambda-grammars from the point of view of multiplicative linear logic. It catalogs the overgeneration and the descriptive inadequacy of lambda-grammars and naive ACGs. Incidentally, Moot introduced what amounts to our hyp-rank 1 restriction, under the name 'strict separation'. Thus hyp-rank is a generalization of strict separation. Since we eschew

the Curry-Howard correspondence for directional types, we also forsake the direct semantic or intuitionistic linear logic interpretation for $\mathcal{AL}_D$ derivations. (Syntax-semantics interface is out of scope for the present paper.)

Kanazawa [4] describes a radically different approach to preventing overgeneration and ensuring the descriptive adequacy of ACGs, based on so-called syntactic features represented by regular constraints, and tree automata that capture those constraints. Nevertheless, there is surprising a similarity: his marking of atomic types by features is similar in spirit to our atomic types like $\langle \bullet, np; s \rangle$ that 'mark', so to speak, the type s with the hypothetical environment containing np.

Retoré and Salvati [12], like us, are interested if ACGs could 'faithfully' represent categorical formalisms, that is, their derivations. There are also many similarities in technical details: our calculus LA and the treatment of lexical entries is similar to the calculus used in their paper, modulo associativity. However, Retoré and Salvati study the non-associative Lambek calculus whereas we use the associative one. Mainly, underlying [12] is the linear lambda calculus. We, in contrast, rely on the algebraic approach and specifically avoid lambda-terms.

5 Discussion

Our approach of representing LGs as algebraic ACGs relies on hyp-rank: the fixed upper bound on the number of not-yet-discharged hypotheses that can appear at any single time in any branch of an L derivation. This section discusses the theoretical significance and the practical insignificance of the hyp-rank. We also say a few words about the development and motivations of the LA calculus.

We should stress that the hyp-rank concerns only those hypotheses of an L derivation that are discharged by the introduction rules in that derivation. The hypotheses that persists until the end (i.e., correspond to the lexicon of LG) do not count towards the hyp-rank. Thus the hyp-rank in no way restricts the size of the LG lexicon. Furthermore, the hyp-rank counts not the total number of hypotheses in a derivation – not the total number of introduction rules – but their maximum number along any single derivation branch. If one branch introduces a hypothesis and then discharges it with an introduction rule, and so does another, independent branch, the hyp-rank of each branch and of the merged derivation is one.

From the practical point of view, the hyp-rank can be disregarded. Whatever large or infinite may be the set of strings in a natural language,

one may expect that recognizing it requires only a bounded, and rather small, number of hypotheses. The success of CCG in parsing natural languages lends credit to this assertion: CCG rules such as composition and lifting are derivable in LG with only one, local assumption. Thus the core CCG (AB grammar plus composition, lifting and associativity) corresponds to LG derivations of hyp-rank one.

From the point of ACG, the fixed hyp-rank qualification can be lifted if one allows polymorphic ACG signatures. On the other hand, it is interesting to investigate classes of context-free languages recognizable by LGs of a given hyp-rank.

The LA calculus was originally developed for a different project: to give some automation to the field of type-logical grammars. The goal is to use the facilities of programming languages to not only mechanically verify the derivations, but also to easily compose them from already checked parts, to reuse in new projects, to develop libraries of derivations and regression testing suites – and to conveniently display derivations and produce figures for papers.

We have used the automation in the present paper. We have embedded LA calculus in OCaml and used the embedding to mechanically check the derivation in Fig.3 and produce the LaTeX code for Figures 1 and 3. In fact, the former was produced from the latter by implementing the proof of Prop.1, which is constructive and can be taken as an algorithm.

6 Conclusions

We have thus demonstrated the strong equivalence of LGs and algebraic ACGs, by exhibiting the construction of an algebraic ACG for a given LG and the hyp-rank. The abstract language of this ACG is the set of LG derivations of the given hyp-rank and the object language is the set of yields of those derivations – with no blow-up in the lexicon.

Contra Moot [9], we conclude that although ACG lack directional types, they are just as descriptively adequate as Lambek Grammars. Thus ACG may, after all, be rightly called categorial grammars.

In the future work we would like to extend our ACG construction to other type logical grammars, such as Hybrid TLG (HTLG).

Acknowledgments We thank Yusuke Kubota for his inspiring challenge. The comments by Richard Moot and the anonymous reviewers are gratefully acknowledged.

This work was partially supported by JSPS KAKENHI Grant Number 17K00091.

References

1. De Groote, P.: Lambek Categorial Grammars as Abstract Categorial Grammars. In: LENLS 13. Logic and Engineering of Natural Language Semantics 13, Tokyo, Japan (Oct 2016), https://hal.inria.fr/hal-01412795
2. de Groote, P.: Towards abstract categorial grammars. In: ACL. pp. 148–155 (2002), http://www.aclweb.org/anthology/P01-1033
3. de Groote, P., Pogodalla, S.: On the expressive power of abstract categorial grammars: Representing context-free formalisms. Journal of Logic, Language and Information **13**(4), 421–438 (2004)
4. Kanazawa, M.: Syntactic features for regular constraints and an approximation of directional slashes in abstract categorial grammars. In: Kubota, Y., Levine, R. (eds.) Proceedings for ESSLLI 2015 Workshop 'Empirical Advances in Categorial Grammar' (CG 2015). pp. 34–70. University of Tsukuba and Ohio State University (2015), https://makotokanazawa.ws.hosei.ac.jp/publications/approx_proc.pdf
5. Kanazawa, M., Pogodalla, S.: Advances in Abstract Categorial Grammars: Language theory and linguistic modeling. Lecture notes, ESSLLI 09. Part 2 (Jul 2009), http://www.loria.fr/equipes/calligramme/acg/publications/esslli-09/2009-esslli-acg-week-2-part-2.pdf
6. Kanazawa, M., Salvati, S.: The string-meaning relations definable by lambek grammars and context-free grammars. In: Morrill, G., Nederhof, M.J. (eds.) FG. Lecture Notes in Computer Science, vol. 8036, pp. 191–208. Springer (2013)
7. Kanazawa, M., Yoshinaka, R.: Lexicalization of Second-Order ACGs. Tech. rep., NII (Jul 2005), https://www.nii.ac.jp/TechReports/public_html/05-012E.html
8. Kubota, Y., Levine, R.: Against ellipsis: Arguments for the direct licensing of 'non-canonical' coordinations. Linguistics and Philosophy **38**(6), 521–576 (2015)
9. Moot, R.: Hybrid type-logical grammars, first-order linear logic and the descriptive inadequacy of lambda grammars (May 26 2014), https://hal.archives-ouvertes.fr/hal-00996724
10. Pentus, M.: Lambek grammars are context free. In: Proc. Eighth Annual Symposium on Logic in Computer Science (LICS '93). pp. 429–433 (1993). https://doi.org/10.1109/LICS.1993.287565
11. Retoré, C.: The Logic of Categorial Grammars: Lecture Notes. Tech. Rep. RR-5703, INRIA (Sep 2005), https://hal.inria.fr/inria-00070313
12. Retoré, C., Salvati, S.: A faithful representation of non-associative lambek grammars in abstract categorial grammars. Journal of Logic, Language and Information **19**(2), 185–200 (2010)
13. Yoshinaka, R., Kanazawa, M.: The complexity and generative capacity of lexicalized abstract categorial grammars. In: Logical Aspects of Computational Linguistics (LACL 2005). Lecture Notes in Computer Science, vol. 3492, pp. 330–346. Springer (2005)