Syntactic Features for Regular Constraints and an Approximation of Directional Slashes in Abstract Categorial Grammars

Makoto Kanazawa*
National Institute of Informatics, Tokyo, Japan, and
SOKENDAI (Graduate University for Advanced Studies)

June 15, 2015

Abstract

I demonstrate the usefulness of syntactic features expressing regular constraints in a linguistic theory based on the formalism of abstract categorial grammars, using a fragment involving *wh*-extraction islands as an example. I then use the same technique to give a method of approximating in ACGs the behavior of the directional slashes of the Lambek calculus.

Keywords: abstract categorial grammar; Lambek calculus; intersection with regular sets; syntactic features

1 Introduction

This paper was motivated by the question of to what extent the formalism of *abstract categorial grammars* (de Groote, 2001) is capable of replicating Lambek grammars, where both formalisms are taken as systems for defining relations between strings and typed λ-terms (understood as representations of meanings). This is a purely technical question about the expressive power of two grammar formalisms, but I believe its resolution to be of interest to linguists since the apparent difficulty of the ACG formalism to simulate Lambek calculus analyses of non-constituent coordination and related phenomena seems to have led some people to conclude that the ACG is fundamentally inadequate as a vehicle to express a satisfactory theory of natural language syntax and compositional semantics.[1]

As a partial answer to this question, this paper presents an automatic procedure to translate an arbitrary Lambek grammar into an ACG that simulates it up to a certain point. This is done by adding two syntactic features to each atomic type, which express

*I am grateful to Chris Tancredi and Greg Kobele for giving me native speakers' judgments.

[1] This view was expressed by Kubota and Levine (2014b, p. 30):

> "...keeping track of the right word order becomes a virtually intractable problem in non-directional variants of CG such as Abstract Categorial Grammar (de Groote 2001) and Lambda Grammar (Muskens 2003)"

and by Moot (2014, p. 2):

> "The abstract categorial grammar treatment suffers from problems of overgeneration and problems at the syntax-semantics interface unlike any other categorial grammar."

regular constraints on the surface positions of "extraction sites". The way the syntactic features are percolated is automatically determined by the regular constraints; this is an application of a general method I developed in Kanazawa (2006). Since each feature has a finite number of possible values, employing it is formally equivalent to splitting each atomic type into a finite number of distinct variants.

Before turning to the simulation of Lambek grammars, I illustrate my general method by showing how λ-abstraction in derivations can be controlled through a finite-valued feature, using a fragment containing *wh*-extraction. (The fragment is vaguely reminiscent of GPSG (Gazdar et al., 1985).) The feature employed in the fragment is similar to what Pogodalla and Pompigne (2012) arrived at by an ad hoc construction. I then proceed to show how one can use the same method to limit λ-abstraction in such a way that the surface position of the extraction site created by λ-abstraction is limited to left or right periphery, mimicking the behavior of the introduction rules for Lambek's directional slashes.

A systematic use of the latter technique results in a procedure to translate an arbitrary Lambek grammar into an ACG. This is not a perfect simulation; the output ACG undergenerates relative to the string-meaning relation of the input Lambek grammar.[2] For the purpose of the description of natural language, this imperfection does not necessarily bode ill for the proposed technique. The discrepancy does not seem to show up in concrete cases that have been discussed by linguists, and I will suggest that there may even be reason to favor the feature mechanism over the directional slashes of the Lambek calculus.

I will start by giving an informal introduction to the ACG formalism, illustrating some of its important properties through concrete examples. Before proceeding to do so in Section 2, however, I would like to make some terminological and conceptual clarifications.

By a *grammar formalism*, I mean a mathematically defined class of finite devices (called *grammars*) equipped with a definition of the *language* of each grammar, which is a set of discrete entities of some sort (strings, trees, λ-terms, pairs of strings and λ-terms, etc.). A grammar formalism is an abstract mathematical object that can and should be studied in isolation from any applications. Each grammar formalism usually distinguishes itself from others by referring to a grammar in its class by a unique compound noun ending in the word "grammar". Thus, a *context-free grammar* or a *CFG* is a grammar belonging to a certain formalism found in any textbook in formal language theory; an *abstract categorial grammar* or an *ACG* is a grammar belonging to the formalism defined by de Groote (2001).[3]

In linguistics, the word "grammar" has another, entirely different use, where it appears as part of a proper name that refers to a particular (usually fairly comprehensive) *linguistic theory* that aims to capture the syntax (and often compositional semantics) of natural language. These names are almost always capitalized. Examples are Lexical-Functional Grammar (LFG), Generalized Phrase Structure Grammar (GPSG), and Head-Driven Phrase Structure Grammar (HPSG). These linguistic theories are often expressed by means of some (usually custom-built) grammar formalism in the sense of the preceding paragraph, but more often than not, the definition of the grammar formalism is

[2] In the initial abstract I submitted to this workshop, I stated that the output ACG also overgenerates relative to the input Lambek grammar. That statement was made in error, and I now think it only undergenerates.

[3] What I call an ACG in this paper is technically a coupling of two ACGs that share the same *abstract signature*. This use of ACGs was called the *transductive paradigm* by de Groote (2001). Other authors have proposed definitions roughly equivalent to de Groote's, most notably Muskens (2003), but not all details are the same.

only implicit in the linguistic theory.[4] (To add to the confusion, these linguistic theories themselves are sometimes called "grammar formalisms" when they serve as frameworks in which to carry out further linguistic investigations.)

The formalism of abstract categorial grammars was introduced without a well-developed linguistic theory to go with it. I think it is fair to say that subsequent uses of the ACG formalism in theoretical and computational linguistics have been sporadic and no broad consensus has emerged as to what form linguistically adequate grammars of English, French, etc., should take within a framework offered by the ACG formalism.[5] In this sense, there is no linguistic theory called *Abstract Categorial Grammar*, at least not yet.[6]

There is another point I wish to make at this point about the relation between a grammar formalism and a linguistic theory. When a grammar formalism is used in a linguistic theory, it need not be the case that the formalism provides the entire metalanguage in which to express individual grammars within that theory. A trivial example is the use of parentheses and braces in phrase structure rules to indicate optional elements and alternatives. These are conventions used to abbreviate multiple phrase structure rules into a single rule and do not appear in the official definition of a phrase structure rule. Nevertheless, as noted by Chomsky (1965, pp. 42–43), these abbreviatory devices may play important roles in expressing linguistically significant generalizations. Another example might be Jacobson's (2014, pp. 94–95) use of "directional rules" in the lexicon of a categorial grammar of English to express the generalization that English is a predominantly *head-first* language. Here, a directional rule is a kind of lexical rule that turns a lexical entry underspecified as to the directions of some slashes into one where the directions of those slashes are specified. Another, particularly elaborate example of an additional mechanism that falls outside of the grammar formalism adopted by a linguistic theory is found in the TAG-based theory of Frank (2002), where the Merge and Move operations are used to generate the finite set of elementary trees of a tree-adjoining grammar.

Now when the workings of such formalism-external devices are specified fully precisely, they may be incorporated into the grammar formalism and the resulting enriched formalism may be studied as a mathematical object. But they need not be. Rather, these devices may simply be viewed as part of a system of notation—compression scheme if you will—that enables compact representations of grammars belonging to the grammar formalism. The choice between these two viewpoints of course does not affect the content of the linguistic theory in any way; it is just a matter of what level of abstraction is the most fruitful in the mathematical study of grammar formalisms. When certain representational devices in the linguistic theory are left out of the grammar formalism, algorithmic

[4] An unfortunate practice that greatly hampers understanding of the relevant linguistic theory, in my opinion. A notable exception in this respect is Tree-Adjoining Grammar (Joshi and Schabes, 1997; Abeillé and Rambow, 2000; Frank, 2002).

[5] Needless to say, examples ("toy grammars") that are used to illustrate properties of a grammar formalism need not constitute any serious attempt to develop a linguistic theory based on that formalism. Also, when a linguist employs a certain formalism in her account of a certain empirical phenomenon, the grammar fragment provided for that purpose may involve choices that are tangential to the points she wants to make and which she would be happy to change in the face of possible criticisms.

[6] In fact, de Groote and Pogodalla (2004, p. 421) go so far as to say

> "Abstract Categorial Grammars are not intended to be yet another grammatical formalism that would compete with other well-established formalisms. They should rather be seen as the kernel of a grammatical framework — in the spirit of (Ranta, 2004) — in which other existing grammatical models may be encoded."

I myself do not share this view. I think the ACG formalism has some unique strengths that make it attractive as a grammar formalism for natural language.

properties of the formalism will guide and inform our understanding of the corresponding properties of the linguistic theory, rather than reveal them completely. This will in no way render the formalism irrelevant to the linguistic theory; on the contrary, such a separation between a grammar formalism and formalism-external representational devices may offer a more effective way to gain important insights into human grammars.

2 Abstract Categorial Grammars

The ACG formalism is based on the *simply typed λ-calculus*, the kind of λ-calculus linguists are familiar with from Montague semantics. *Simple types* are formed from atomic types with the connective →; if A and B are simple types, then $A \to B$ is a simple type. A *λ-term* is either a variable, a constant, an application MN of λ-terms M and N, or a λ-abstract $\lambda x.M$, where M is a λ-term and x is a variable. Note that I write MN instead of $M(N)$ for applications; the former notation is the standard one in λ-calculus. (We of course use parentheses where they are necessary to remove ambiguity.) Application is assumed to be left-associative, so MNP means $(MN)P$, not $M(NP)$. A sequence of λs is abbreviated to one, as in $\lambda xyz.x(yz) = \lambda x.\lambda y.\lambda z.x(yz)$. Another standard abbreviation is employed in writing types: $\alpha \to \beta \to \gamma$ abbreviates $\alpha \to (\beta \to \gamma)$ (i.e., the connective → associates to the right). I write $\alpha^n \to \beta$ for $\alpha \to \cdots \to \alpha \to \beta$, with n occurrences of α. I assume that the reader is familiar with the notions of *free* and *bound* variables, with λ-conversion or *β-reduction*, and with how types are assigned to λ-terms given the types of variables and constants.

Roughly, an *abstract categorial grammar* (ACG)[7] consists of a finite set of triples of the form (α, M, N), where α is a simple type over some finite set $\mathcal{P}$ of atomic types, and M and N are closed λ-terms of types $\sigma(\alpha)$ and $\tau(\alpha)$, respectively, where σ and τ are the two *type substitutions* specified by the grammar. In this paper, I call the triples (α, M, N) comprising an ACG *grammar entries*. The first component α of a grammar entry is a *syntactic type*. The λ-terms M and N are the λ-terms in the *form dimension* and the *meaning dimension*, respectively. The former is usually assumed to be a *linear* λ-term, while the latter is not restricted in that way.[8] These λ-terms may contain constants as well as bound variables and λ. In this paper, the constants appearing in the form dimension are all of the same type *str*, which is the type of string, except the constant ∘, which is of type $str \to str \to str$ and stands for string concatenation. We write ∘ as an infix operator. The constants appearing in the meaning dimension have various types built from atomic types (e, t, etc.).

An ACG generates a pair consisting of a string and a λ-term by combining grammar entries through application and linear λ-abstraction in accordance with their syntactic types. A succinct formal definition of ACGs is found in the appendix (Section A).

2.1 ACG Encoding of a CFG

Let us see our first example of an ACG, which is based on the translation from CFGs to ACGs given by de Groote (2001).

Consider the context-free grammar in Figure 1, where each rule is associated with a λ-term that specifies how the meaning of an expression built with that rule is determined by

[7] See footnote 3.

[8] A λ-term is *linear* if each λ binds exactly one occurrence of a variable. The restriction to linear λ-terms can be relaxed to *almost linear* λ-terms (Kanazawa, 2007, 2011, 2014) without affecting the computational properties of ACGs.

$$
\begin{array}{rlrl}
s \to np\ vp & [\![vp]\!]\,[\![np]\!] & np \to \mathsf{Leslie} & \mathbf{Leslie}^{e} \\
vp \to v1\ np & \lambda x^{e}.[\![v1]\!]\,[\![np]\!]\,x & np \to \mathsf{Robin} & \mathbf{Robin}^{e} \\
vp \to v2\ s\_bar & \lambda x^{e}.[\![v2]\!]\,[\![s\_bar]\!]\,x & np \to \mathsf{Terry} & \mathbf{Terry}^{e} \\
vp \to v3\ s\_int & \lambda x^{e}.[\![v3]\!]\,[\![s\_int]\!]\,x & v1 \to \mathsf{hates} & \lambda y^{e} x^{e}.\mathbf{hate}^{e\to e\to t}\,y\,x \\
s\_bar \to \mathsf{that}\ s & [\![s]\!] & v1 \to \mathsf{likes} & \lambda y^{e} x^{e}.\mathbf{like}^{e\to e\to t}\,y\,x \\
s\_int \to \mathsf{whether}\ s & \mathbf{yn}^{t\to q}\,[\![s]\!] & v2 \to \mathsf{thinks} & \lambda y^{t} x^{e}.\mathbf{think}^{t\to e\to t}\,y\,x \\
 & & v2 \to \mathsf{claims} & \lambda y^{t} x^{e}.\mathbf{claim}^{t\to e\to t}\,y\,x \\
 & & v3 \to \mathsf{wonders} & \lambda y^{q} x^{e}.\mathbf{wonder}^{q\to e\to t}\,y\,x \\
 & & v3 \to \mathsf{knows} & \lambda y^{q} x^{e}.\mathbf{know}^{q\to e\to t}\,y\,x
\end{array}
$$

Figure 1: A CFG with Montague semantics.

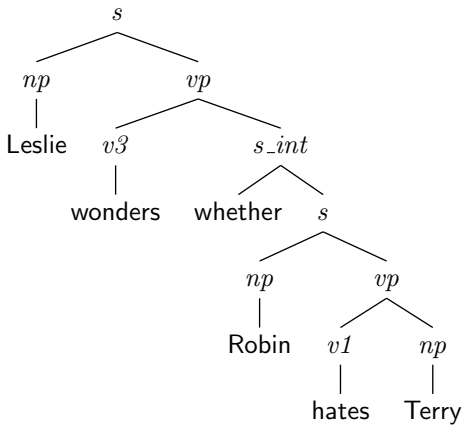


Figure 2: A CFG derivation tree.

the meanings of its immediate constituents. I use lower-case italic letters for nonterminals (i.e., syntactic categories) and sans serif for terminals (i.e., words). The type of a constant or variable is indicated by a superscript at its first occurrence. Here, e is the type of individual, t is the type of proposition, q is the type of question, and the constant $\mathbf{yn}$ denotes a function that turns a proposition into a yes-no (polar) question. According to this context-free grammar, the string Leslie wonders whether Robin hates Terry has the derivation tree in Figure 2 and its meaning is calculated to be

$$
\begin{aligned}
&((\lambda y^{q} x^{e}.\mathbf{wonder}^{q\to e\to t}\,y\,x)(\mathbf{yn}^{t\to q}\,(((\lambda y^{e} x^{e}.\mathbf{hate}^{e\to e\to t}\,y\,x)\,\mathbf{Terry}^{e})\,\mathbf{Robin}^{e})))\,\mathbf{Leslie}^{e} \\
&\qquad = \mathbf{wonder}^{q\to e\to t}\,(\mathbf{yn}^{t\to q}\,(\mathbf{hate}^{e\to e\to t}\,\mathbf{Terry}^{e}\,\mathbf{Robin}^{e}))\,\mathbf{Leslie}^{e}. \quad (1)
\end{aligned}
$$

Figure 3 lists the grammar entries of the ACG that encodes the CFG in Figure 1. Each rule of the CFG corresponds to an entry of the ACG. The correspondence should be fairly obvious. When a CFG has a rule $X \to w_0 X_1 w_1 \dots X_n w_n$, where $X, X_1, \dots, X_n$ are nonterminals and $w_0, w_1, \dots, w_n$ are strings of terminals, the corresponding ACG entry has the syntactic type $X_1 \to \dots \to X_n \to X$ and the λ-term $\lambda z_1^{str} \dots z_n^{str}.w_0 \circ z_1 \circ w_1 \circ \dots \circ z_n \circ w_n$ in the form dimension. The λ-term in the meaning dimension of the entry is obtained from the λ-term attached to the CFG rule by replacing $[\![X_1]\!], \dots, [\![X_n]\!]$ with appropriately typed variables $z_1, \dots, z_n$ and abstracting over them.

The derivation trees of the ACG are almost isomorphic to the derivation trees of the CFG. Each node of an ACG derivation tree is licensed by a grammar entry. Figure 4

$$
\begin{array}{lll}
(\ np \to vp \to s, & \lambda z_1^{str} z_2^{str}.z_1 \circ z_2, & \lambda z_1^{e} z_2^{e\to t}.z_2 z_1 \ ) \\
(\ v1 \to np \to vp, & \lambda z_1^{str} z_2^{str}.z_1 \circ z_2, & \lambda z_1^{e\to e\to t} z_2^{e} x^{e}.z_1 z_2 x \ ) \\
(\ v2 \to s\_bar \to vp, & \lambda z_1^{str} z_2^{str}.z_1 \circ z_2, & \lambda z_1^{t\to e\to t} z_2^{t} x^{e}.z_1 z_2 x \ ) \\
(\ v3 \to s\_int \to vp & \lambda z_1^{str} z_2^{str}.z_1 \circ z_2, & \lambda z_1^{q\to e\to t} z_2^{q} x^{e}.z_1 z_2 x \ ) \\
(\ s \to s\_bar, & \lambda z_1^{str}.\mathsf{that} \circ z_1, & \lambda z_1^{t}.z_1 \ ) \\
(\ s \to s\_int, & \lambda z_1^{str}.\mathsf{whether} \circ z_1, & \lambda z_1^{t}.\mathbf{yn}^{t\to q}\, z_1 \ ) \\
(\ np, & \mathsf{Leslie}, & \mathbf{Leslie}^{e} \ ) \\
(\ np, & \mathsf{Robin}, & \mathbf{Robin}^{e} \ ) \\
(\ np, & \mathsf{Terry}, & \mathbf{Terry}^{e} \ ) \\
(\ v1, & \mathsf{hates}, & \lambda y^{e} x^{e}.\mathbf{hate}^{e\to e\to t}\, y\, x \ ) \\
(\ v1, & \mathsf{likes}, & \lambda y^{e} x^{e}.\mathbf{like}^{e\to e\to t}\, y\, x \ ) \\
(\ v2, & \mathsf{thinks}, & \lambda y^{t} x^{e}.\mathbf{think}^{t\to e\to t}\, y\, x \ ) \\
(\ v2, & \mathsf{claims}, & \lambda y^{t} x^{e}.\mathbf{claim}^{t\to e\to t}\, y\, x \ ) \\
(\ v3, & \mathsf{wonders}, & \lambda y^{q} x^{e}.\mathbf{wonder}^{q\to e\to t}\, y\, x \ ) \\
(\ v3, & \mathsf{knows}, & \lambda y^{q} x^{e}.\mathbf{know}^{q\to e\to t}\, y\, x \ )
\end{array}
$$

Figure 3: The ACG encoding of the CFG in Figure 1.

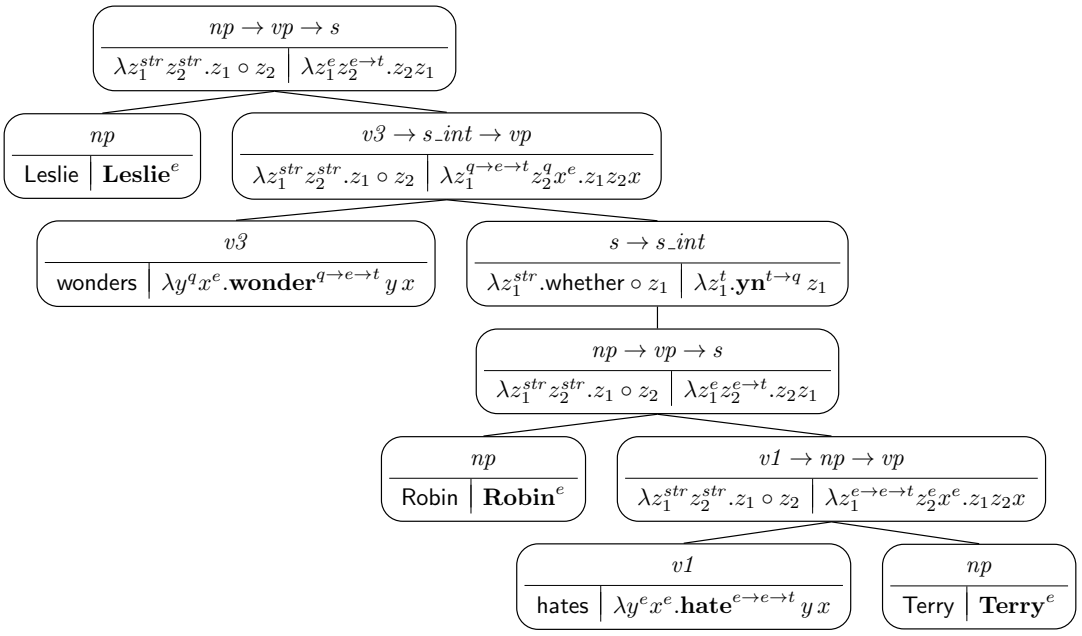


Figure 4: An ACG derivation tree representing a CFG derivation tree.

shows the ACG derivation tree corresponding to the CFG derivation tree in Figure 2. The well-formedness of an ACG derivation tree is checked by calculating the type of each subtree. Thus, a subtree T whose root node is labeled by a grammar entry of the form $(\alpha_1 \to \cdots \to \alpha_n \to p, M, N)$ is well-formed and is assigned type p if it has n immediate subtrees that are assigned types $\alpha_1, \ldots, \alpha_n$, respectively. The form/meaning associated with T is the result of applying the λ-term in the form/meaning dimension of the grammar entry $(\alpha_1 \to \cdots \to \alpha_n \to p, M, N)$ to the forms/meanings associated with the immediate subtrees of T. The λ-term in the form dimension associated with the entire derivation tree in Figure 4 is (after β-reduction) calculated to be

$$\mathsf{Leslie} \circ (\mathsf{wonders} \circ (\mathsf{whether} \circ (\mathsf{Robin} \circ (\mathsf{hates} \circ \mathsf{Terry})))),$$

and the corresponding λ-term in the meaning dimension is calculated to be (1) above. These λ-terms are guaranteed to be well-formed because of the existence of type substitutions σ and τ such that for every grammar entry (α, M, N), M is a well-formed λ-term of type $\sigma(\alpha)$ and N is a well-formed λ-term of type $\tau(\alpha)$. In the case of this ACG, σ maps each atomic syntactic type to *str*, and τ is the following type substitution:

$$\begin{aligned} s &\mapsto t, & v1 &\mapsto e \to e \to t, \\ np &\mapsto e, & v2 &\mapsto t \to e \to t, \\ vp &\mapsto e \to t, & v3 &\mapsto q \to e \to t. \\ s\_bar &\mapsto t, \\ s\_int &\mapsto q, \end{aligned}$$

The ACG in Figure 3 is particularly simple in two respects. First, the syntactic type of a grammar entry is always of the form $p_1 \to \cdots \to p_n \to p$ with $n \geq 0$, where $p_1, \ldots, p_n, p$ are all atomic types. ACGs with this property are (somewhat misleadingly) called *second-order* ACGs (see Kanazawa, 2009).[9] Second, in this ACG, the type substitution σ maps every atomic type to the type *str*. The next example is different in that σ maps some atomic syntactic types to complex types.

2.2 ACG encoding of a TAG

This example uses the method of encoding tree-adjoining grammars into ACGs due to de Groote (2002) and is loosely based on an example used by Pogodalla (2009, Chapter 2).

The tree-adjoining grammar in Figure 5 generates a fragment that is similar to the one generated by the CFG in Figure 1, but extended with *wh*-questions. For example, this TAG generates (2) through the derivation tree in Figure 6.

(2) Leslie wonders who Robin thinks that Terry hates

An interesting property of this TAG is that long distance *wh*-extraction must be mediated by bridge verbs that occur in auxiliary trees like the one for thinks.[10] Consequently, sentence (3) below is not generated. (Note that this is just an illustrative example.)

(3) Leslie wonders who Robin knows whether Terry hates

The ACG encoding of the TAG in Figure 5 coupled with a suitable semantics is given in Figure 7. We have a new constant $\mathbf{who}^{(e\to t)\to q}$ in the meaning dimension that turns a unary property into a *wh*-question. In this ACG, all atomic types are mapped to *str* by

[9] Second-order ACGs are roughly the same as what Kracht (2003) called *de Saussure grammars*.

[10] This treatment of *wh*-extraction in the TAG formalism is originally due to Kroch (1987). The account is slightly modified here for simplicity.

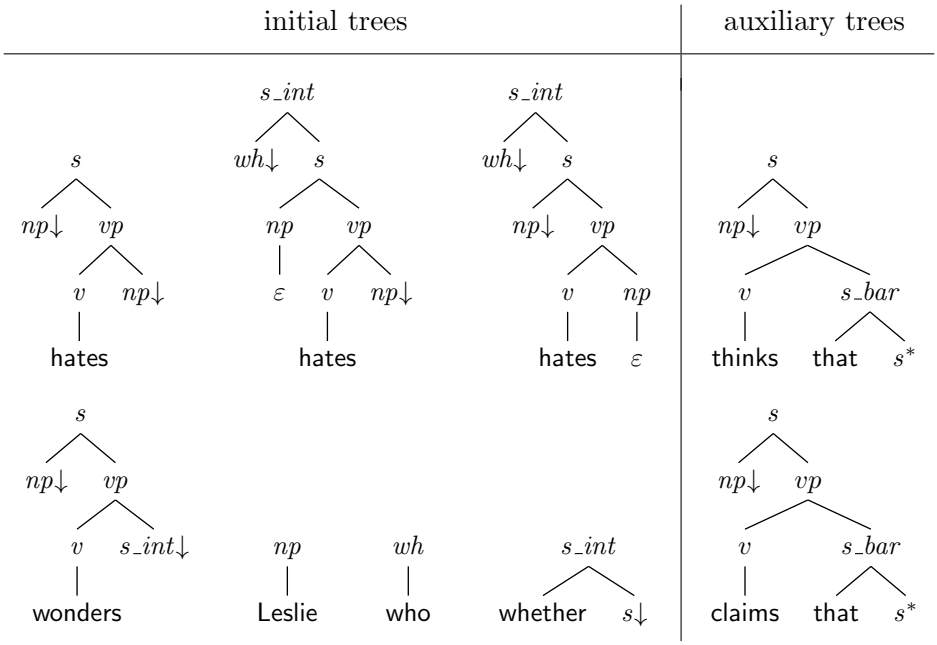


Figure 5: A tree-adjoining grammar. The arrow ↓ indicates substitution nodes and the asterisk ∗ indicates foot nodes of auxiliary trees. Each of the two auxiliary trees can *adjoin* into any node labeled with s (without ↓ or ∗). Three initial trees with likes (similar to those with hates), one with knows (similar to that with wonders), and one each with Robin and Terry (similar to that with Leslie) are not shown.

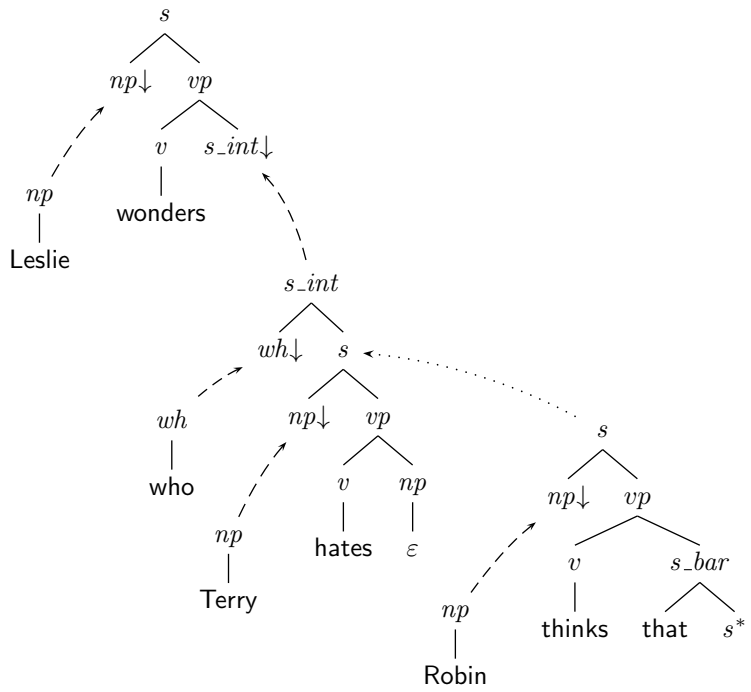


Figure 6: A TAG derivation tree. The dashed arrows indicate substitution and the dotted arrow adjunction.

$(\ np \to np \to s_A \to s,$	$\lambda z_1^{str} z_2^{str} z_3^{str \to str}.z_3(z_2 \circ (\mathsf{hates} \circ z_1)),$	$\lambda z_1^e z_2^e z_3^{t \to t}.\quad z_3(\mathbf{hate}^{e \to e \to t}\, z_1\, z_2)\ )$
$(\ np \to wh \to s_A \to s\_int,$	$\lambda z_1^{str} z_2^{str} z_3^{str \to str}.z_2 \circ (z_3(\varepsilon \circ (\mathsf{hates} \circ z_1))),$	$\lambda z_1^e z_2^{(e \to t) \to q} z_3^{t \to t}.\ z_2(\lambda x^e.z_3(\mathbf{hate}^{e \to e \to t}\, z_1\, x))\ )$
$(\ np \to wh \to s_A \to s\_int,$	$\lambda z_1^{str} z_2^{str} z_3^{str \to str}.z_2 \circ (z_3(z_1 \circ (\mathsf{hates} \circ \varepsilon))),$	$\lambda z_1^e z_2^{(e \to t) \to q} z_3^{t \to t}.\ z_2(\lambda x^e.z_3(\mathbf{hate}^{e \to e \to t}\, x\, z_1))\ )$
$(\ s\_int \to np \to s_A \to s,$	$\lambda z_1^{str} z_2^{str} z_3^{str \to str}.z_3(z_2 \circ (\mathsf{wonders} \circ z_1)),$	$\lambda z_1^q z_2^e z_3^{t \to t}.z_3(\mathbf{wonder}^{q \to e \to t}\, z_1\, z_2)\ )$
$(\ s\_int \to np \to s_A \to s,$	$\lambda z_1^{str} z_2^{str} z_3^{str \to str}.z_3(z_2 \circ (\mathsf{knows} \circ z_1)),$	$\lambda z_1^q z_2^e z_3^{t \to t}.z_3(\mathbf{know}^{q \to e \to t}\, z_1\, z_2)\ )$
$(\ np,$	$\mathsf{Leslie},$	$\mathbf{Leslie}^e\ )$
$(\ wh,$	$\mathsf{who},$	$\lambda y^{e \to t}\mathbf{who}^{(e \to t) \to q}(\lambda x^e.yx)\ )$
$(\ s \to s\_int,$	$\lambda z_1^{str}.\mathsf{whether} \circ z_1,$	$\lambda z_1^t.\mathbf{yn}^{t \to q}\, z_1\ )$
$(\ np \to s_A \to s_A,$	$\lambda z_1^{str} z_2^{str} x^{str}.z_2(z_1 \circ (\mathsf{thinks} \circ (\mathsf{that} \circ x))),$	$\lambda z_1^e z_2^{t \to t} x^t.z_2(\mathbf{think}\, x\, z_1)\ )$
$(\ np \to s_A \to s_A,$	$\lambda z_1^{str} z_2^{str} x^{str}.z_2(z_1 \circ (\mathsf{claims} \circ (\mathsf{that} \circ x))),$	$\lambda z_1^e z_2^{t \to t} x^t.z_2(\mathbf{claim}\, x\, z_1)\ )$
$(\ s_A,$	$\lambda x^{str}.x,$	$\lambda x^t.x\ )$

Figure 7: The ACG encoding of the TAG in Figure 5 coupled with a Montague semantics. There is one grammar entry in the ACG for each elementary tree of the TAG. The last entry in the ACG does not correspond to any elementary tree of the TAG and expresses the optionality of adjunction.

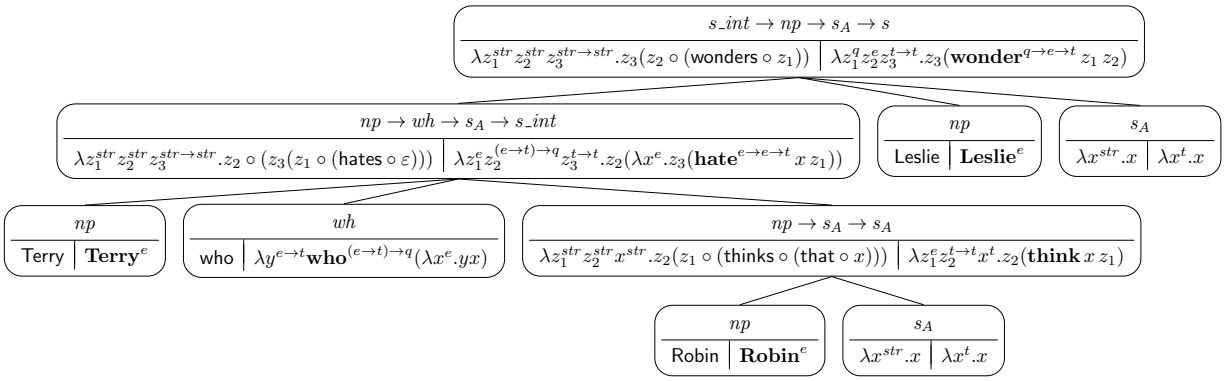


Figure 8: An ACG derivation tree representing a TAG derivation tree.

the type substitution σ except s_A, which is mapped to $str \to str$. The type substitution τ for the meaning dimension is as follows:

$$\begin{aligned} s &\mapsto t, \\ np &\mapsto e, \\ s\_int &\mapsto q, \\ wh &\mapsto (e \to t) \to q, \\ s_A &\mapsto t \to t. \end{aligned}$$

Figure 8 shows the ACG derivation tree for the pair

$$\Big(\mathsf{Leslie} \circ (\mathsf{wonders} \circ (\mathsf{who} \circ (\mathsf{Robin} \circ (\mathsf{thinks} \circ (\mathsf{that} \circ (\mathsf{Terry} \circ (\mathsf{hates} \circ \varepsilon))))))),$$
$$\mathbf{wonder}^{q \to e \to t}\, (\mathbf{who}^{(e \to t) \to q}\, (\lambda x^e.\mathbf{think}^{t \to e \to t}\, (\mathbf{hate}^{e \to e \to t}\, x\, \mathbf{Terry}^e)\, \mathbf{Robin}^e))\, \mathbf{Leslie}^e \Big).$$

Note that all entries of this ACG except the last contains at least one constant in the form dimension. An ACG all of whose entries have this property is said to be *lexicalized*.[11]

[11]This terminology comes from the TAG formalism, where a TAG is said to be *lexicalized* (Joshi and Schabes, 1997) if each of its elementary trees contains a terminal symbol.

$$\begin{array}{llll}
(\ wh \to (np \to s) \to q, & \lambda z_1^{str} z_2^{str \to str}.z_1 \circ (z_2\,\varepsilon), & \lambda z_1^{(e\to t)\to q} z_2^{e\to t}.z_1(\lambda x^e.z_2 x) &) \\
(\ wh & \mathsf{who} & \lambda y^{e\to t}.\mathbf{who}^{(e\to t)\to q}\,(\lambda x^e.yx) &) \\
(\ s \to s \to s, & \lambda z_1^{str} z_2^{str}.z_2 \circ (\mathsf{and} \circ z_1), & \lambda z_1^t z_2^t.\ \wedge^{t\to t\to t}\ z_1\, z_2 &) \\
(\ vp \to vp \to vp, & \lambda z_1^{str} z_2^{str}.z_2 \circ (\mathsf{and} \circ z_1), & \lambda z_1^{e\to t} z_2^{e\to t} x^e.\ \wedge^{t\to t\to t}\ (z_1 x)\,(z_2 x) &)
\end{array}$$

Figure 9: Additional entries for the ACG fragment handling *wh*-extraction and coordination.

It is easy to turn this ACG into a lexicalized one that generates the same set of string-meaning pairs.[12]

2.3 Encoding Freewheeling Extraction by Lambda Abstraction

The previous two examples of ACGs were both second-order ACGs. The derivations in a second-order ACG form a *local* set of trees,[13] and in this respect, second-order ACGs, together with well-known mildly context-sensitive grammars like TAGs, are "context-free" grammars in a generalized sense. Our next example is a higher-order ACG and extends the ACG in Section 2.1 by using λ-abstraction in derivations to handle extraction. We also add entries for coordination for good measure. The entries of this ACG are those in Figure 9 in addition to those in Figure 3. The new meaning constant $\wedge^{t\to t\to t}$ is the truth function for conjunction written in prefix notation. The type substitutions σ and τ for this ACG extend those for the ACG in Figure 3 by

$$\sigma(wh) = str, \quad \tau(wh) = (e \to t) \to q.$$

With a higher-order ACG, derivations are no longer trees (in general). They may contain nodes labeled by bound variables and λ-operators. (The binding relation between these nodes could be represented by arcs, so the structure of a derivation is a kind of graph.) We notate a bound variable node with an oval and a λ node with a diamond. The whole derivation will be a closed λ-term made up of grammar entries, bound variables, and λ. Each λ must bind exactly one occurrence of a bound variable; i.e., a derivation must be a linear λ-term. The form and meaning associated with a derivation are computed by interpreting λ-abstraction by λ-abstraction, replacing the type α of bound variable x^α with $\sigma(\alpha)$ and $\tau(\alpha)$, respectively.

Here are a few string-meaning pairs generated by this ACG:

$$\Big(\ \mathsf{Leslie} \circ (\mathsf{wonders} \circ (\mathsf{who} \circ (\mathsf{Terry} \circ (\mathsf{hates} \circ \varepsilon)))),\ \mathbf{wonder}^{q\to e\to t}\,(\mathbf{who}^{(e\to t)\to q}\,(\lambda x^e.\mathbf{hate}^{e\to e\to t}\,x\,\mathbf{Terry}^e))\,\mathbf{Leslie}^e\ \Big) \tag{4}$$

$$\Big(\ \mathsf{Leslie} \circ (\mathsf{wonders} \circ (\mathsf{who} \circ (\mathsf{Robin} \circ (\mathsf{thinks} \circ (\mathsf{that} \circ (\mathsf{Terry} \circ (\mathsf{hates} \circ \varepsilon))))))),\ \mathbf{wonder}^{q\to e\to t}\,(\mathbf{who}^{(e\to t)\to q}\,(\lambda x^e.\mathbf{think}^{t\to e\to t}\,(\mathbf{hate}^{e\to e\to t}\,x\,\mathbf{Terry}^e)\,\mathbf{Robin}^e))\,\mathbf{Leslie}^e\ \Big) \tag{5}$$

$$\Big(\ \mathsf{Leslie} \circ (\mathsf{wonders} \circ (\mathsf{who} \circ (\varepsilon \circ (\mathsf{likes} \circ \mathsf{Robin})))),\ \mathbf{wonder}^{q\to e\to t}\,(\mathbf{who}^{(e\to t)\to q}\,(\lambda x^e.\mathbf{like}^{e\to e\to t}\,\mathbf{Robin}^e\,x))\ \Big) \tag{6}$$

[12] It is known that every second-order ACG can be converted to an equivalent lexicalized second-order ACG (Kanazawa and Yoshinaka, 2005), but this generally makes σ a more complex type substitution.

[13] A set of trees is *local* if membership of a tree in the set depends only on the "local" fragments of the tree consisting of a node together with all its children (see Comon et al., 2008).

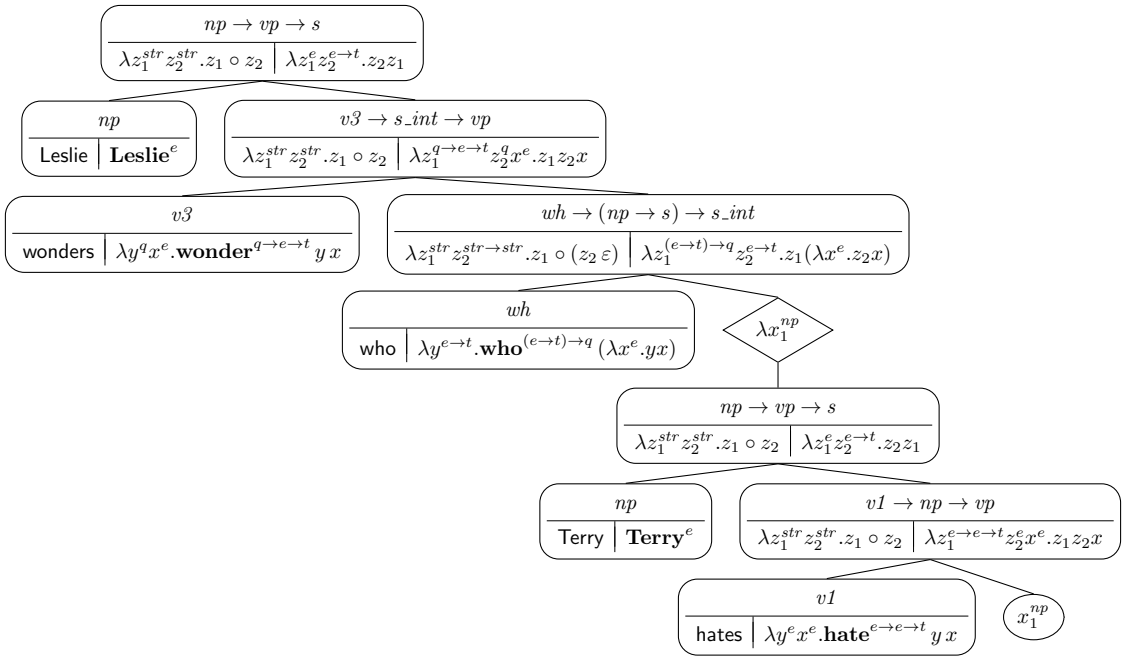


Figure 10: An ACG derivation λ-term that is not a tree.

The derivation for (4) is shown in Figure 10.

This ACG allows *wh*-extraction freely within the constraint imposed by linearity of derivations. In addition to pairs like (4)–(6), string-meaning pairs like the following are generated:

$$\Big(\mathsf{Leslie} \circ (\mathsf{wonders} \circ (\mathsf{who} \circ (\mathsf{Robin} \circ (\mathsf{thinks} \circ (\mathsf{that} \circ (\varepsilon \circ (\mathsf{hates} \circ \mathsf{Terry})))))))),$$
$$\mathbf{wonder}^{q\to e\to t}\,(\mathbf{who}^{(e\to t)\to q}\,(\lambda x^e.\mathbf{think}^{t\to e\to t}\,(\mathbf{hate}^{e\to e\to t}\,\mathbf{Terry}^e\,x)\,\mathbf{Robin}^e))\,\mathbf{Leslie}^e\Big) \tag{7}$$

$$\Big(\mathsf{Leslie} \circ (\mathsf{wonders} \circ (\mathsf{who} \circ (\mathsf{Robin} \circ (\mathsf{knows} \circ (\mathsf{whether} \circ (\mathsf{Terry} \circ (\mathsf{hates} \circ \varepsilon)))))))),$$
$$\mathbf{wonder}^{q\to e\to t}\ (\mathbf{who}^{(e\to t)\to q}\,(\lambda x^e.\mathbf{know}^{q\to e\to t}\,(\mathbf{yn}^{t\to q}\,(\mathbf{hate}^{e\to e\to t}\,x\,\mathbf{Terry}^e))\,\mathbf{Robin}^e))\ \mathbf{Leslie}^e\Big) \tag{8}$$

$$\Big(\mathsf{Leslie} \circ (\mathsf{wonders} \circ (\mathsf{who} \circ ((\mathsf{Robin} \circ (\mathsf{likes} \circ \varepsilon)) \circ (\mathsf{and} \circ (\mathsf{Terry} \circ (\mathsf{hates} \circ \mathsf{Robin})))))),$$
$$\mathbf{wonder}^{q\to e\to t}\ (\mathbf{who}^{(e\to t)\to q}\,(\lambda x^e.\,\wedge^{t\to t\to t}\,(\mathbf{like}^{e\to e\to t}\,x\,\mathbf{Robin}^e)\,(\mathbf{hates}^{e\to e\to t}\,\mathbf{Robin}^e\,\mathbf{Terry}^e)))\ \mathbf{Leslie}^e\Big) \tag{9}$$

In Section 3, I show how one can constrain λ-abstraction to block sentences like (8) and (9).

2.4 Some Important Properties of ACGs

We have seen three examples of ACGs. The first two (Sections 2.1 and 2.2) are second-order ACGs faithfully encoding a CFG and a TAG, respectively. No λ-abstraction occurs in

the derivations of these grammars. Neither is lexicalized, but the second ACG is nearly so and is easily converted to a lexicalized ACG.[14] The second ACG maps an atomic syntactic type s_A (corresponding to the auxiliary trees of the encoded TAG) to a nonatomic type $str \to str$. This is necessary to simulate the operation of adjunction in TAGs.

Formal properties of second-order ACGs are by now fairly well-understood (see Kanazawa, 2007, 2011, 2009). In particular, their string-generating power is the same as that of *multiple context-free grammars* (Seki et al., 1991) or *linear context-free rewriting systems* (Vijay-Shanker et al., 1987).[15]

Our third example (Section 2.3) is a higher-order ACG in that it contains a grammar entry with a syntactic type that is not of the form $p_1 \to \cdots \to p_n \to p$ (with $p_1, \ldots, p_n, p$ atomic). Compared to second-order ACGs, higher-order ACGs are of very high complexity[16] and their formal properties are not very well-understood. Although the ACG in Section 2.3 is not lexicalized, it can easily be converted to a *semilexicalized* ACG by combining the first two entries in Figure 9.[17]

It is important to remember that neither lexicalization nor semilexicalization is required of ACGs in general. This is one of several crucial differences that sets ACGs apart from categorial grammars like Lambek grammars, where every entry is attached to an overt word.

We have used constants $\circ$ and ε to represent the operation of string concatenation and the empty string, respectively. Alternatively, we can view them as uninterpreted symbols; the form component of the generated pairs will then be trees of a certain kind, rather than strings.[18] This shift in perspective will be useful in the next section.

3 Syntactic Features for Regular Constraints

Traditionally, various *island constraints* have been posited to account for unacceptable instances of *wh*-extraction. Thus, the whether-clause in (8) creates a *whether island*, and (9) involves a *coordinate structure island*.[19] In this section, I show how a general construction I gave in Kanazawa (2006) automatically supplies a finite-valued syntactic feature that captures island effects. A similar use of a syntactic feature to control *wh*-extraction is found in Pogodalla and Pompigne (2012). My point here is that whenever one can express a relevant constraint in terms of a regular set of trees (or strings), a syntactic feature that captures that constraint is automatically obtained.

[14] Since all second-order ACGs can be lexicalized, it is possible to lexicalize the ACG in Section 2.1 as well. In general, lexicalization of a second-order ACG requires a grammar transformation similar to conversion to Greibach normal form (Kanazawa and Yoshinaka, 2005).

[15] This was first proved by Salvati (2007).

[16] It has recently been shown that the emptiness and universal recognition problems for higher-order ACGs are of non-elementary complexity (Lazić and Schmitz, 2014); it is not known whether they are decidable. In comparison, the emptiness and universal recognition problems for second-order ACGs are P-complete and in EXPTIME (but at least PSPACE-hard), respectively (Kanazawa, 2011).

[17] An ACG is called *semilexicalized* (Salvati, 2005) if entries with higher-order syntactic types contain at least one constant in the form dimension. Yoshinaka (2006) showed that semilexicalized ACGs can be lexicalized. The universal recognition problem for semilexicalized ACGs is decidable (Salvati, 2005).

[18] We can even give a general definition of ACGs in such a way that they generate pairs of (not necessarily binary) trees and λ-terms. The string-meaning pairs of the grammar will then be obtained by taking the yields of the trees in the generated pairs.

[19] The grammatical status of these purported island violations has been in dispute in recent years (see, e.g., Phillips, 2006, 2013; Hofmeister and Sag, 2010; Chaves, 2012). To the extent that the apparent unacceptability of *wh*-extraction from these "islands" is due to extra-grammatical factors, my case for the usefulness of syntactic features to control extraction is weakened. Here, I assume that at least some purported islands are grammatical in nature.

Let us take the ACG in Section 2.3 (call it G) and treat $\circ$ and ε as uninterpreted symbols. The ACG generates pairs of binary trees and λ-term representations of meanings. The symbol ε occupies the positions of *wh*-extraction sites; they may be thought of as traces of *wh*-movement. In order to express the fact that certain constructions are islands, we introduce two unary function symbols I, O of type $str \to str$ and modify the form dimension of some of the entries as follows:

$$\begin{aligned}
&(\ s \to s_int,\ \lambda z_1^{str}.\mathsf{whether} \circ z_1,\ \dots\ ) \rightsquigarrow (\ s \to s_int,\ \lambda z_1^{str}.\,\mathsf{I}\,(\mathsf{whether} \circ z_1),\ \dots\ )\\
&(\ wh \to (np \to s) \to q,\ \lambda z_1^{str} z_2^{str\to str}.z_1 \circ (z_2\,\varepsilon),\ \dots\ )\\
&\qquad\qquad \rightsquigarrow (\ wh \to (np \to s) \to s_int,\ \lambda z_1^{str} z_2^{str\to str}.\,\mathsf{I}\,(z_1 \circ \mathsf{O}\,(z_2\,\varepsilon)),\ \dots\ )\\
&(\ s \to s \to s,\ \lambda z_1^{str} z_2^{str}.z_2 \circ (\mathsf{and} \circ z_1),\ \dots\ ) \rightsquigarrow (\ s \to s \to s,\ \lambda z_1^{str} z_2^{str}.\,\mathsf{I}\,(z_2 \circ (\mathsf{and} \circ z_1)),\ \dots\ )\\
&(\ vp \to vp \to vp,\ \lambda z_1^{str} z_2^{str}.z_2 \circ (\mathsf{and} \circ z_1),\ \dots\ )\\
&\qquad\qquad \rightsquigarrow (\ vp \to vp \to vp,\ \lambda z_1^{str} z_2^{str}.\,\mathsf{I}\,(z_2 \circ (\mathsf{and} \circ z_1)),\ \dots\ )
\end{aligned}$$

The symbol I serves to mark islands for *wh*-extraction, and the symbol O marks the "scope" of fronted *wh*-phrases. (We think of an occurrence of ε inside the scope of a *wh*-phrase as "bound" by that *wh*-phrase.) Call the modified ACG G'. The form components of some of the generated pairs now look as follows:

$$\mathsf{Leslie} \circ (\mathsf{wonders} \circ \mathsf{I}\left(\mathsf{who} \circ \mathsf{O}\left(\mathsf{Terry} \circ (\mathsf{hates} \circ \varepsilon)\right)\right)) \tag{4′}$$

$$\mathsf{Leslie} \circ (\mathsf{wonders} \circ \mathsf{I}\left(\mathsf{who} \circ \mathsf{O}\left(\mathsf{Robin} \circ (\mathsf{thinks} \circ (\mathsf{that} \circ (\mathsf{Terry} \circ (\mathsf{hates} \circ \varepsilon))))\right)\right)) \tag{5′}$$

$$\mathsf{Leslie} \circ (\mathsf{wonders} \circ \mathsf{I}\left(\mathsf{who} \circ \mathsf{O}\left(\varepsilon \circ (\mathsf{likes} \circ \mathsf{Robin})\right)\right)) \tag{6′}$$

$$\mathsf{Leslie} \circ (\mathsf{wonders} \circ \mathsf{I}\left(\mathsf{who} \circ \mathsf{O}\left(\mathsf{Robin} \circ (\mathsf{knows} \circ \mathsf{I}\left(\mathsf{whether} \circ (\mathsf{Terry} \circ (\mathsf{hates} \circ \varepsilon))\right)\right)\right)\right)) \tag{8′}$$

$$\mathsf{Leslie} \circ (\mathsf{wonders} \circ \mathsf{I}\left(\mathsf{who} \circ \mathsf{O}\left(\mathsf{I}\left((\mathsf{Robin} \circ (\mathsf{likes} \circ \varepsilon)) \circ (\mathsf{and} \circ (\mathsf{Terry} \circ (\mathsf{hates} \circ \mathsf{Robin})))\right)\right)\right)) \tag{9′}$$

The forms (4′)–(6′) obey the island constraints and (8′)–(9′) don't. Figure 11 shows (5′) and (8′) in graphical notation.

The trees that do and those that don't obey island constraints can be distinguished by the following deterministic *bottom-up finite tree automaton* (see, e.g., Comon et al., 2008), which has just two states, [GAP −] ("contains no unbound gap") and [GAP +] ("contains an unbound gap"):

$$\begin{aligned}
\varepsilon &\to [\textsc{gap}\,+],\\
a &\to [\textsc{gap}\,-] \quad \text{for each terminal symbol } a,\\
[\textsc{gap}\,-] \circ [\textsc{gap}\,-] &\to [\textsc{gap}\,-],\\
[\textsc{gap}\,-] \circ [\textsc{gap}\,+] &\to [\textsc{gap}\,+],\\
[\textsc{gap}\,+] \circ [\textsc{gap}\,-] &\to [\textsc{gap}\,+],\\
\mathsf{I}\,[\textsc{gap}\,-] &\to [\textsc{gap}\,-],\\
\mathsf{O}\,[\textsc{gap}\,+] &\to [\textsc{gap}\,-].
\end{aligned}$$

Note that there is no transition for $[\textsc{gap}\,+] \circ [\textsc{gap}\,+]$. The automaton accepts a tree if it can be rewritten to $[\textsc{gap}\,-]$ by applying these transitions bottom-up. The accepted trees are those such that

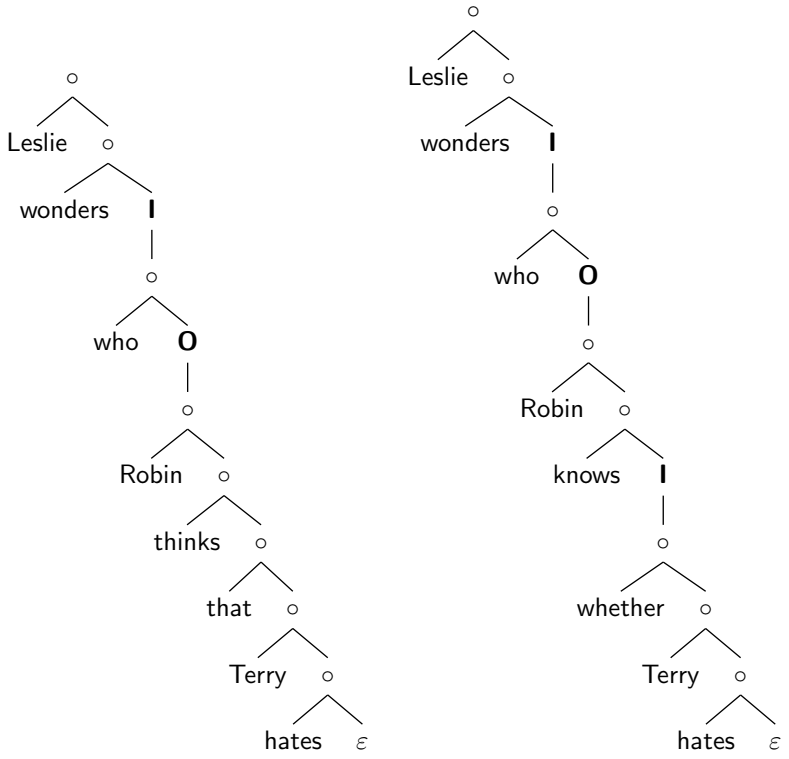


Figure 11: A tree that obeys island constraints (left) and one that does not (right).

- no subtree contains more than one unbound gap,
- no subtree that is an island (marked by **I**) contains any unbound gap, and
- the entire tree contains no unbound gap.

It is important to note that this characterization of island constraints in terms of a regular set of trees is possible no matter what construction creates an island, provided that a fronted *wh*-phrase (together with its scope) creates an island of its own.[20]

We modified the ACG G in Section 2.3 into an ACG G' which uses new constants **I** and **O** in the form dimension and which allows a simple characterization of the effects of islands in terms of a regular set R of trees (represented in the form of a finite tree automaton). By a construction of Kanazawa (2006, Lemma 3.3 and Theorem 5.1), we can form an ACG G'' such that

- G'' generates (M, N) if and only if G' generates (M, N) and $M \in R$,
- G' is the image of G'' under a type substitution that maps atomic syntactic types of G'' to atomic syntactic types of G'.

I do not give a formal definition of this construction here, but merely illustrate it with examples. An atomic type of G'' is of the form $p[\textsc{gap} \pm]$, where p is an atomic type of G'.[21]

[20]This is why Pogodalla and Pompigne (2012) needed a much more complex, infinite-valued feature mechanism to capture the fact that a tensed clause creates a scope island. The scope of a quantifier may contain a variable that is bound by another quantifier higher up, so quantification does not itself create a scope island.

[21]In the general case, this is a little more complex. If $\sigma(p)$ contained k occurrences of *str* instead of just one, then an atomic type of G'' that gets mapped to p would have k copies of the GAP feature attached to p.

Each grammar entry (α, M, N) of G' gives rise to zero, one, or more grammar entries of the form (α', M, N) such that α is the result of erasing all the feature specifications from α', and α' "induces" a typing of M consistent with the tree automaton for R. Let us use the entry

$$(\ wh \to (np \to s) \to s\_int,\ \lambda z_1^{str} z_2^{str \to str}.\, \mathsf{I}\,(z_1 \circ \mathsf{O}\,(z_2\,\varepsilon)),\ \lambda z_1^{(e\to t)\to q} z_2^{e\to t}.z_1(\lambda x^e.z_2 x)\ )$$

to illustrate this. Consider

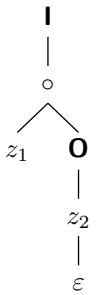
$$wh[\textsc{gap}\ g_1] \to (np[\textsc{gap}\ g_2] \to s[\textsc{gap}\ g_3]) \to s\_int[\textsc{gap}\ g_4],$$

where $g_1, g_2, g_3, g_4 \in \{-, +\}$, as a feature-specified version of the syntactic type of the above entry. Using g_1, g_2, g_3, we add new transitions

$$z_1 \to [\textsc{gap}\ g_1],$$
$$z_2\,[\textsc{gap}\ g_2] \to [\textsc{gap}\ g_3]$$

to the tree automaton and run it on the input tree

$\mathsf{I}\,(z_1 \circ \mathsf{O}\,(z_2\,\varepsilon)) =$



(the "body" of the λ-term in the form dimension of the entry) to see if it can be rewritten to $[\textsc{gap}\ g_4]$. Since ε gets rewritten to $[\textsc{gap}\ +]$, we must have $g_2 = +$. Since the only transition for O is $\mathsf{O}\,[\textsc{gap}\ +] \to [\textsc{gap}\ -]$, we must have $g_3 = +$. Likewise, since the only transition for I is $\mathsf{I}\,[\textsc{gap}\ -] \to [\textsc{gap}\ -]$, we must have $g_4 = -$ for the tree to be rewritten to $[\textsc{gap}\ g_4]$. Since we must have $[\textsc{gap}\ g_1] \circ [\textsc{gap}\ -] \to [\textsc{gap}\ -]$, it follows that g_1 must be $-$. So the only possibility is $(g_1, g_2, g_3, g_4) = (-, +, +, -)$.[22]

The entire grammar G'' is given in Figure 12. (We write $[-]$ and $[+]$ instead of $[\textsc{gap}\ -]$ and $[\textsc{gap}\ +]$ to save space.)

Finally, erasing I and O from G'' and restoring the original λ-terms in the form dimension gives a grammar G''' that generates the desired subset of the form-meaning pairs generated by G. The method is summarized in Figure 13.[23]

[22] This corresponds to the alternative typing

$$\lambda z_1^{[\textsc{gap}\ -]} z_2^{[\textsc{gap}\ +]\to[\textsc{gap}\ +]}.\, \mathsf{I}^{[\textsc{gap}\ -]\to[\textsc{gap}\ -]}\,(z_1 \circ^{[\textsc{gap}\ -]\to[\textsc{gap}\ -]\to[\textsc{gap}\ -]} \mathsf{O}^{[\textsc{gap}\ +]\to[\textsc{gap}\ -]}\,(z_2\,\varepsilon^{[\textsc{gap}\ +]}))$$

of the λ-term in the form dimension of the entry.

[23] There remains the problem of blocking (7) while allowing sentences like (6) and Leslie wonders who Robin thinks hates Terry. This could be done, for example, by letting the $np \to vp \to s$ entry specify subject as an island and positing special entries for *in-situ wh*-subjects and for bridge verbs combining with a vp, à la GPSG (Gazdar et al., 1985).

It is also possible to add *across-the-board wh*-extraction to the present fragment by using a three-valued feature $[\textsc{gap}\ -]$, $[\textsc{gap}\ +]$ ("properly contains a gap"), $[\textsc{gap}\ =]$ ("is a gap") instead and adding the following entry to G', where E is a new unary function symbol that requires its argument to be empty:

$$(\,(np \to s) \to (np \to s) \to np \to s,\quad \lambda z_1^{str\to str} z_2^{str\to str} x^{str}.\ \mathsf{I}\,(\mathsf{O}\,(z_2\,\varepsilon) \circ (\mathsf{and} \circ \mathsf{O}\,(z_1\,\varepsilon))) \circ (\mathsf{E}\,x),\quad \lambda z_1^{e\to t} z_2^{e\to t} x^e.\ \wedge^{t\to t\to t}\ (z_1 x)\,(z_2 x)\,)$$

(The position of $\mathsf{E}\,x$ may look unnatural, but what matters is the final product, a grammar with none of these markers.)

$(\ np[g_1] \to vp[g_2] \to s[g_3],$	$\lambda z_1^{str} z_2^{str}.z_1 \circ z_2,$	$\lambda z_1^{e} z_2^{e\to t}.z_2\, z_1$ $)$
$(\ v1[g_1] \to np[g_2] \to vp[g_3],$	$\lambda z_1^{str} z_2^{str}.z_1 \circ z_2,$	$\lambda z_1^{e\to e\to t} z_2^{e} x^{e}.z_1\, z_2 x$ $)$
$(\ v2[g_1] \to s\_bar[g_2] \to vp[g_3],$	$\lambda z_1^{str} z_2^{str}.z_1 \circ z_2,$	$\lambda z_1^{t\to e\to t} z_2^{t} x^{e}.z_1\, z_2 x$ $)$
$(\ v3[g_1] \to s\_int[g_2] \to vp[g_3]$	$\lambda z_1^{str} z_2^{str}.z_1 \circ z_2,$	$\lambda z_1^{q\to e\to t} z_2^{q} x^{e}.z_1\, z_2 x$ $)$
$(\ s[g] \to s\_bar[g],$	$\lambda z_1^{str}.\mathsf{that} \circ z_1,$	$\lambda z_1^{t}.z_1$ $)$
$(\ s[-] \to s\_int[-],$	$\lambda z_1^{str}.\mathsf{I}\,(\mathsf{whether} \circ z_1),$	$\lambda z_1^{t}.\mathbf{yn}^{t\to q}\, z_1$ $)$
$(\ np[-],$	Leslie,	$\mathbf{Leslie}^{e}$ $)$
$(\ np[-],$	Robin,	$\mathbf{Robin}^{e}$ $)$
$(\ np[-],$	Terry,	$\mathbf{Terry}^{e}$ $)$
$(\ v1[-],$	hates,	$\lambda y^{e} x^{e}.\mathbf{hate}^{e\to e\to t}\, y\, x$ $)$
$(\ v1[-],$	likes,	$\lambda y^{e} x^{e}.\mathbf{like}^{e\to e\to t}\, y\, x$ $)$
$(\ v2[-],$	thinks,	$\lambda y^{t} x^{e}.\mathbf{think}^{t\to e\to t}\, y\, x$ $)$
$(\ v2[-],$	claims,	$\lambda y^{t} x^{e}.\mathbf{claim}^{t\to e\to t}\, y\, x$ $)$
$(\ v3[-],$	wonders,	$\lambda y^{q} x^{e}.\mathbf{wonder}^{q\to e\to t}\, y\, x$ $)$
$(\ v3[-],$	knows,	$\lambda y^{q} x^{e}.\mathbf{know}^{q\to e\to t}\, y\, x$ $)$
$(\ wh[-] \to (np[+] \to s[+]) \to s\_[-],$	$\lambda z_1^{str} z_2^{str\to str}.\mathsf{I}\,(z_1 \circ \mathsf{O}\,(z_2\, \varepsilon)),$	$\lambda z_1^{(e\to t)\to q} z_2^{e\to t}.z_1(\lambda x^{e}.z_2 x)$ $)$
$(\ wh[-]$	who	$\lambda y^{e\to t}.\mathbf{who}^{(e\to t)\to q}\,(\lambda x^{e}.y x)$ $)$
$(\ s[-] \to s[-] \to s[-],$	$\lambda z_1^{str} z_2^{str}.\mathsf{I}\,(z_2 \circ (\mathsf{and} \circ z_1)),$	$\lambda z_1^{t} z_2^{t}.\ \wedge^{t\to t\to t}\ z_1\, z_2$ $)$
$(\ vp[-] \to vp[-] \to vp[-],$	$\lambda z_1^{str} z_2^{str}.\mathsf{I}\,(z_2 \circ (\mathsf{and} \circ z_1)),$	$\lambda z_1^{e\to t} z_2^{e\to t} x^{e}.\ \wedge^{t\to t\to t}\ (z_1 x)\,(z_2 x)$ $)$

Figure 12: An ACG with island and *wh*-scope markers that respects island constraints. Here, $(g_1, g_2, g_3) \in \{(-,-,-),(-,+,+),(+,-,+)\}$ and $g \in \{-,+\}$.

G		G'		G''		G'''
with unconstrained *wh*-extraction	$\rightsquigarrow$	with island and *wh*-scope markers	$\rightsquigarrow$	with markers respecting islands	$\rightsquigarrow$	without markers respecting islands

Figure 13: Deriving an ACG with a feature that respects *wh*-extraction islands.

Two remarks about the island and *wh*-scope markers I, O are in order. First, these markers are devices that we employed to *discover* an appropriate syntactic feature to use; they are *not* part of the final grammar that is obtained by this method. The "derivation" of the grammar G''' (Figure 13) has nothing to do with the derivations of individual sentences generated by G'''. Second, the present example is simple enough that it's easy to come up with a suitable feature mechanism without the help of these markers. The advantage of using them is that the grammar automatically comes with a correctness proof. The use of these markers reduces the linguist's job to the specification of the constraint in terms of a regular set of trees. As long as this specification is correct, the grammar is guaranteed to be correct.

4 Directional Slashes and Syntactic Features

The most popular style of linguistic analysis based on the formalism of ACGs has been the one exemplified by the fragment in Muskens (2003), which may be loosely described as "non-directionalization" of a Lambek grammar. Moot (2014) quite rightly pointed out that this approach does not scale well; once you add constructions like coordination, adverbial modification, etc., that are usually treated by Lambek grammars using complex functional types that take functional types as arguments, it seems that constraints on word order imposed by directional slashes of the Lambek calculus cannot be replicated by any amount of tweaking in the form dimension of the ACG written in this style. Needless to say, this does not make the formalism of ACGs "descriptively inadequate". Clearly, other styles of analysis, perhaps incorporating some ideas from TAGs, are possible within the

ACG formalism. Even the CFG-based style of analysis we have used in Sections 2.1 and 2.3 is immune to some of the criticisms of Moot (2014), namely, the ones based on adverbial modification and coordination of standard constituents. However, non-constituent coordination, a traditional stronghold of categorial grammars, does seem to present a difficulty for any ACG analysis that treats non-standard constituents as resulting from extraction (modeled by λ-abstraction).

I illustrate the problem in Section 4.1 using Right Node Raising, still following the same style of analysis from Sections 2.1 and 2.3. The solution I propose is to use a finite-valued syntactic feature again. A systematic use of this technique leads to a translation from an *arbitrary* Lambek grammar into an ACG with features (Section 4.2). In Section 4.3, I apply the technique to the treatment of Gapping by Kubota and Levine (2014a,b), which uses their *hybrid type-logical grammar* formalism. I end with my skepticism about treating Gapping with left- and right-peripheral extractions, suggesting that Lambek's directional slashes lack the flexibility one needs to control the extraction sites involved in Gapping (Section 4.4).

4.1 Right Node Raising

In Lambek calculus, Right Node Raising has been analyzed as coordination of two expressions of type s/X (for some type X). For example, sentence (10) would involve assigning the conjunction and the type $((s/np)\backslash(s/np))/(s/np)$:

(10) Terry hates and Leslie likes Robin.

A naive attempt to translate this entry for and into an ACG grammar entry would result in either of the following:

$$\begin{array}{l}(\,(np \to s) \to (np \to s) \to np \to s,\\ \quad \lambda z_1^{str\to str} z_2^{str\to str} x^{str}.((z_2\,\varepsilon) \circ (\mathsf{and} \circ (z_1\,\varepsilon))) \circ x,\ \lambda z_1^{e\to t} z_2^{e\to t} x^{e}.\ \wedge^{t\to t\to t}\ (z_1\,x)\,(z_2\,x)\,)\end{array} \tag{11}$$

$$\begin{array}{l}(\,(np \to s) \to (np \to s) \to np \to s,\\ \quad \lambda z_1^{str\to str} z_2^{str\to str} x^{str}.(z_2\,\varepsilon) \circ (\mathsf{and} \circ (z_1\,x)),\ \lambda z_1^{e\to t} z_2^{e\to t} x^{e}.\ \wedge^{t\to t\to t}\ (z_1\,x)\,(z_2\,x)\,)\end{array} \tag{12}$$

As pointed out by Moot (2014), adopting either of these entries in an ACG would result in overgeneration. If we add (11) to the fragment in Section 2.1, the following pair (with the meaning "Terry hates Robin and Robin likes Leslie") will be generated, with the derivation in Figure 14:

$$\begin{array}{l}(\,((\mathsf{Terry} \circ (\mathsf{hates} \circ \varepsilon)) \circ (\mathsf{and} \circ (\varepsilon \circ (\mathsf{likes} \circ \mathsf{Leslie})))) \circ \mathsf{Robin},\\ \quad \wedge^{t\to t\to t}\ (\mathbf{like}^{e\to e\to t}\,\mathbf{Leslie}^{e}\,\mathbf{Robin}^{e})\,(\mathbf{hate}^{e\to e\to t}\,\mathbf{Robin}^{e}\,\mathbf{Terry}^{e})\,).\end{array}$$

If we add (12) instead, then the following pair (with the same meaning) will be generated with a similar derivation:

$$\begin{array}{l}(\,(\mathsf{Terry} \circ (\mathsf{hates} \circ \varepsilon)) \circ (\mathsf{and} \circ (\mathsf{Robin} \circ (\mathsf{likes} \circ \mathsf{Leslie}))),\\ \quad \wedge^{t\to t\to t}\ (\mathbf{like}^{e\to e\to t}\,\mathbf{Leslie}^{e}\,\mathbf{Robin}^{e})\,(\mathbf{hate}^{e\to e\to t}\,\mathbf{Robin}^{e}\,\mathbf{Terry}^{e})\,).\end{array}$$

In the former approach, ungrammatical strings are generated, while in the latter approach, grammatical strings are paired with incorrect meanings.

Let us concentrate on the first approach (11) and see if we can remedy it. The Lambek entry for and requires each of the conjunct in Right Node Raising to contain an np gap

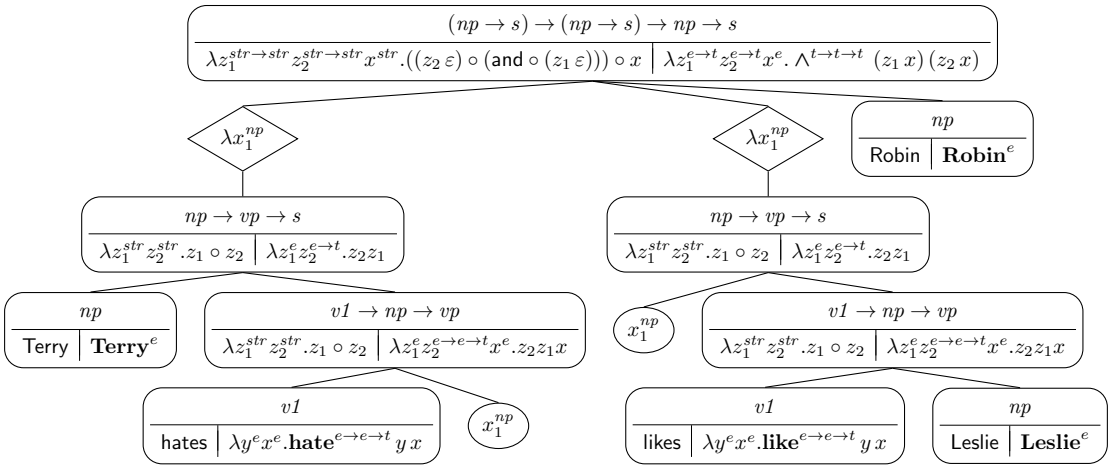


Figure 14: A derivation of Terry hates and likes Leslie Robin with the meaning "Terry hates Robin and Robin likes Leslie".

at its right periphery, but the ACG entry (11) cannot enforce that requirement. Since the requirement concerns the positions of the empty string symbol ε in the form component of generated pairs, we can try to use a syntactic feature to filter out unwanted sentences, like we did with violations of island constraints on *wh*-extraction.

We need to enforce that a gap occur at the right edge of each conjunct in a Right Node Raising construction. Since this is not a requirement on gaps in general, we have to distinguish gaps "bound" by the RNR entry for and from gaps bound by, e.g., *wh*-phrases. For this purpose, we introduce a special symbol ε^R for a right-peripheral gap, and mark the scope of its binder by $\mathbf{O}^R$. Note that these are markers we temporarily introduce to automatically discover a suitable feature mechanism; these are part of the specification, not the delivered product. The λ-term in the form dimension of the entry (11) is modified with these markers as follows:[24]

$$\begin{aligned}&(\,(np \to s) \to (np \to s) \to np \to s,\\ &\qquad \lambda z_1^{str \to str}\, z_2^{str \to str}\, x^{str}.(\mathbf{O}^R\,(z_2\,\varepsilon^R) \circ (\mathsf{and} \circ \mathbf{O}^R\,(z_1\,\varepsilon^R))) \circ x,\ \ldots\,)\end{aligned} \tag{13}$$

With this change, the derivation in Figure 14 now generates the following pair:

$$\begin{aligned}&(\,(\mathbf{O}^R\,(\mathsf{Terry} \circ (\mathsf{hates} \circ \varepsilon^R)) \circ (\mathsf{and} \circ \mathbf{O}^R\,(\varepsilon^R \circ (\mathsf{likes} \circ \mathsf{Leslie})))) \circ \mathsf{Robin},\\ &\qquad \wedge^{t \to t \to t}\ (\mathbf{like}^{e \to e \to t}\,\mathbf{Leslie}^e\,\mathbf{Robin}^e)\,(\mathbf{hate}^{e \to e \to t}\,\mathbf{Robin}^e\,\mathbf{Terry}^e)\,).\end{aligned}$$

We can express the constraint that needs to be satisfied as follows:

- Any occurrence of ε^R must be the rightmost leaf of a subtree marked by $\mathbf{O}^R$.

The deterministic bottom-up finite tree automaton that captures this constraint again has just two states, [RGAP 0] and [RGAP 1], which indicate the absence/presence of a right-peripheral gap. Its transitions are as follows:

$$\begin{aligned}\varepsilon^R &\to [\text{RGAP } 1],\\ a &\to [\text{RGAP } 0] \quad \text{for each terminal symbol } a,\\ [\text{RGAP } 0] \circ [\text{RGAP } 0] &\to [\text{RGAP } 0],\\ [\text{RGAP } 0] \circ [\text{RGAP } 1] &\to [\text{RGAP } 1],\\ \mathbf{O}^R\,[\text{RGAP } 1] &\to [\text{RGAP } 0].\end{aligned} \tag{14}$$

[24] We assume $\mathbf{O}^R$ binds stronger than $\circ$, so that $\mathbf{O}^R\,x \circ y$ means $(\mathbf{O}^R\,x) \circ y$.

$$
\begin{array}{lll}
(\ np[0] \to vp[r] \to s[r], & \lambda z_1^{str}\, z_2^{str}.z_1 \circ z_2, & \lambda z_1^{e}\, z_2^{e\to t}.z_2 z_1 \ ) \\
(\ v1[0] \to np[r] \to vp[r], & \lambda z_1^{str}\, z_2^{str}.z_1 \circ z_2, & \lambda z_1^{e\to e\to t}\, z_2^{e} x^{e}.z_1 z_2 x \ ) \\
(\ v2[0] \to s\_bar[r] \to vp[r], & \lambda z_1^{str}\, z_2^{str}.z_1 \circ z_2, & \lambda z_1^{t\to e\to t}\, z_2^{t} x^{e}.z_1 z_2 x \ ) \\
(\ v3[0] \to s\_int[r] \to vp[r] & \lambda z_1^{str}\, z_2^{str}.z_1 \circ z_2, & \lambda z_1^{q\to e\to t}\, z_2^{q} x^{e}.z_1 z_2 x \ ) \\
(\ s[r] \to s\_bar[r], & \lambda z_1^{str}.\mathsf{that} \circ z_1, & \lambda z_1^{t}.z_1 \ ) \\
(\ s[r] \to s\_int[r], & \lambda z_1^{str}.\mathsf{whether} \circ z_1, & \lambda z_1^{t}.\mathbf{yn}^{t\to q}\, z_1 \ ) \\
(\ np[0], & \mathsf{Leslie}, & \mathbf{Leslie}^{e} \ ) \\
(\ np[0], & \mathsf{Robin}, & \mathbf{Robin}^{e} \ ) \\
(\ np[0], & \mathsf{Terry}, & \mathbf{Terry}^{e} \ ) \\
(\ v1[0], & \mathsf{hates}, & \lambda y^{e} x^{e}.\mathbf{hate}^{e\to e\to t}\, y\, x \ ) \\
(\ v1[0], & \mathsf{likes}, & \lambda y^{e} x^{e}.\mathbf{like}^{e\to e\to t}\, y\, x \ ) \\
(\ v2[0], & \mathsf{thinks}, & \lambda y^{t} x^{e}.\mathbf{think}^{t\to e\to t}\, y\, x \ ) \\
(\ v2[0], & \mathsf{claims}, & \lambda y^{t} x^{e}.\mathbf{claim}^{t\to e\to t}\, y\, x \ ) \\
(\ v3[0], & \mathsf{wonders}, & \lambda y^{q} x^{e}.\mathbf{wonder}^{q\to e\to t}\, y\, x \ ) \\
(\ v3[0], & \mathsf{knows}, & \lambda y^{q} x^{e}.\mathbf{know}^{q\to e\to t}\, y\, x \ ) \\
(\ (np[1] \to s[1]) \to & \lambda z_1^{str\to str}\, z_2^{str\to str} x^{str}. & \lambda z_1^{e\to t}\, z_2^{e\to t} x^{e}.\ \wedge^{t\to t\to t}\ (z_1\, x)\, (z_2\, x) \ ) \\
\quad (np[1] \to s[1]) \to np[r] \to s[r], & \quad (\mathbf{O}^{R}\, (z_2\, \varepsilon^{R}) \circ (\mathsf{and} \circ \mathbf{O}^{R}\, (z_1\, \varepsilon^{R}))) \circ x, &
\end{array}
$$

Figure 15: An ACG with markers ε^R and $\mathbf{O}^R$ containing an entry for Right Node Raising. Here, $r \in \{0, 1\}$, and we write [0] and [1] for [RGAP 0] and [RGAP 1].

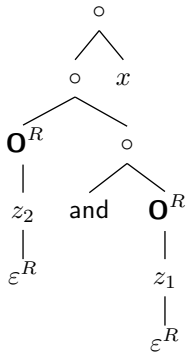
Using these transitions, we can calculate the features to assign to the occurrences of atomic types in the syntactic type of the entry (13). In order for a feature-specified syntactic type

$$(np[\textsc{rgap}\, r_1] \to s[\textsc{rgap}\, r_2]) \to (np[\textsc{rgap}\, r_3] \to s[\textsc{rgap}\, r_4]) \to np[\textsc{rgap}\, r_5] \to s[\textsc{rgap}\, r_6]$$

to be legitimate, the tree automaton (14) augmented with the transitions

$$
\begin{aligned}
z_1\, [\textsc{rgap}\, r_1] &\to [\textsc{rgap}\, r_2], \\
z_2\, [\textsc{rgap}\, r_3] &\to [\textsc{rgap}\, r_4], \\
x &\to [\textsc{rgap}\, r_5],
\end{aligned}
$$

must accept the tree



with final state [RGAP r_6]. We see that we must have $(r_1, r_2) = (r_3, r_4) = (1, 1)$, and $r_5 = r_6$, obtaining a new entry

$$(np[\textsc{rgap}\, 1] \to s[\textsc{rgap}\, 1]) \to (np[\textsc{rgap}\, 1] \to s[\textsc{rgap}\, 1]) \to np[\textsc{rgap}\, r] \to s[\textsc{rgap}\, r] \qquad \text{where } r \in \{0, 1\}.$$

Doing the same with the entries in Figure 3 results in the ACG in Figure 15. Note that the "final product" ACG is the result of removing $\mathbf{O}^R$ and changing ε^R to ε in these entries.

I leave it as an exercise for the reader to combine the fragment in Figure 15 with the one in Figure 12. This will require making a few decisions. You need to complete each of

$$
\begin{array}{lll}
(\, np, & \textsf{Leslie} &) \\
(\, np \to np \to s, & \lambda z_1^{str} z_2^{str}. z_2 \circ (\textsf{likes} \circ z_1) &) \\
(\, (np \to s) \to s, & \lambda z_1^{str\to str}.\textsf{only\_she} \circ \mathbf{O}^L\, (z_1\, \varepsilon^L) &) \\
(\, (np \to s) \to s, & \lambda z_1^{str\to str}.\, \mathbf{O}^R\, (z_1\, \varepsilon^R) \circ \textsf{only\_him} &)
\end{array}
$$

$$
\begin{array}{l}
(\, (np \to np \to s) \to (np \to np \to s) \to np \to np \to s, \\
\qquad \lambda z_1^{str\to str\to str} z_2^{str\to str\to str} z_3^{str} z_4^{str}. \\
\qquad\quad z_4 \circ ((\mathbf{O}^R\, (\mathbf{O}^L\, (z_2\, \varepsilon^R\, \varepsilon^L)) \circ (\textsf{and} \circ \mathbf{O}^R\, (\mathbf{O}^L\, (z_1\, \varepsilon^R\, \varepsilon^L)))) \circ z_3)\,) \\
(\, (((np \to s) \to s) \to s) \to (((np \to s) \to s) \to s) \to ((np \to s) \to s) \to s, \\
\qquad \lambda z_1^{((str\to str)\to str)\to str} z_2^{((str\to str)\to str)\to str} z_3^{(str\to str)\to str}. \\
\qquad (\mathbf{O}^R\, (z_2(\lambda y^{str\to str}.\, \mathbf{O}^R\, (y\, \varepsilon^R) \circ \varepsilon^R)) \circ (\textsf{and} \circ \mathbf{O}^R\, (z_1(\lambda y^{str\to str}.\, \mathbf{O}^R\, (y\, \varepsilon^R) \circ \varepsilon^R)))) \circ \\
\qquad\qquad \mathbf{O}^L\, (z_3(\lambda x^{str}.\varepsilon^L \circ x))\,)
\end{array}
$$

Figure 16: An ACG translation of a Lambek grammar with markers. Not shown are entries for Robin and Terry similar to that for Leslie, and an entry for hates similar to that for likes.

the two tree automata with transitions for markers used in the other fragment and possibly add markings to the form components of some of the entries. Once these decisions are made, however, the rest of the grammar writing is automatic.

4.2 Translating Lambek grammars into ACGs

The use of the RGAP feature in the preceding section can be generalized to a method of translating an arbitrary Lambek grammar into an ACG (with finite-valued syntactic features). I only sketch the method here. The precise definition is found in the appendix (Section A).

Let us consider the following Lambek grammar as an example (we ignore the meaning dimension since it is simply preserved in the ACG):

$$
\begin{aligned}
np &: \textsf{Leslie}, \textsf{Robin}, \textsf{Terry} \\
tv = (np\backslash s)/np &: \textsf{likes}, \textsf{hates} \\
subj = s/(np\backslash s) &: \textsf{only\_she} \\
obj = (s/np)\backslash s &: \textsf{only\_him} \\
(tv\backslash tv)/tv &: \textsf{and} \\
((s/obj)\backslash(s/obj))/(s/obj) &: \textsf{and}
\end{aligned}
\tag{15}
$$

The last entry is used for sentences like Leslie likes and Robin hates only_him paired with the meaning "Leslie likes only him and Robin hates only him" (with "him" deictic). The other entry for and is for transitive verb coordination as in Leslie likes and hates Terry.

Translating this Lambek grammar into an ACG with markers in a way analogous to (13) gives the ACG in Figure 16 (the meaning dimension is omitted). Here, $\mathbf{O}^L$ and ε^L are the analogues of $\mathbf{O}^R$ and ε^R for left-peripheral gaps; in a well-formed sentence, any occurrence of ε^L must be the leftmost leaf of some subtree marked with $\mathbf{O}^L$. This constraint is expressed by the following tree automaton with two states, [LGAP 0] and [LGAP 1]:

$$
\begin{aligned}
\varepsilon^L &\to [\text{LGAP}\,1], \\
\varepsilon^R &\to [\text{LGAP}\,0], \\
a &\to [\text{LGAP}\,0] \quad \text{for each terminal symbol } a,
\end{aligned}
$$

$$
\begin{array}{ll}
(\ np[0,0], & \mathsf{Leslie} \qquad\qquad)\\
(\ np[0,r] \to np[l,0] \to s[l,r], & \lambda z_1^{str} z_2^{str}.z_2 \circ (\mathsf{likes} \circ z_1) \qquad)\\
(\ (np[1,0] \to s[1,r]) \to s[0,r], & \lambda z_1^{str\to str}.\mathsf{only\_she} \circ (z_1\,\varepsilon) \qquad)\\
(\ (np[0,1] \to s[l,1]) \to s[l,0], & \lambda z_1^{str\to str}.(z_1\,\varepsilon) \circ \mathsf{only\_him} \qquad)
\end{array}
$$

$$
\begin{array}{l}
(\ (np[0,1] \to np[1,0] \to s[1,1]) \to (np[0,1] \to np[1,0] \to s[1,1]) \to np[0,r] \to np[l,0] \to s[l,r],\\
\qquad \lambda z_1^{str\to str\to str} z_2^{str\to str\to str} z_3^{str} z_4^{str}.z_4 \circ (((z_2\,\varepsilon\,\varepsilon) \circ (\mathsf{and} \circ (z_1\,\varepsilon\,\varepsilon))) \circ z_3)\ )\\
(\ (((np[0,1] \to s[l_1,1]) \to s[l_1,1]) \to s[0,1]) \to\\
\quad (((np[0,1] \to s[l_2,1]) \to s[l_2,1]) \to s[l_3,1]) \to ((np[0,r_1] \to s[1,r_1]) \to s[1,r_2]) \to s[l_3,r_2],\\
\qquad \lambda z_1^{((str\to str)\to str)\to str} z_2^{((str\to str)\to str)\to str} z_3^{(str\to str)\to str}.\\
\qquad\quad ((z_2(\lambda y^{str\to str}.(y\,\varepsilon) \circ \varepsilon)) \circ (\mathsf{and} \circ (z_1(\lambda y^{str\to str}.(y\,\varepsilon) \circ \varepsilon)))) \circ z_3(\lambda x^{str}.\varepsilon \circ x)\ )
\end{array}
$$

Figure 17: An ACG translation of a Lambek grammar with features.

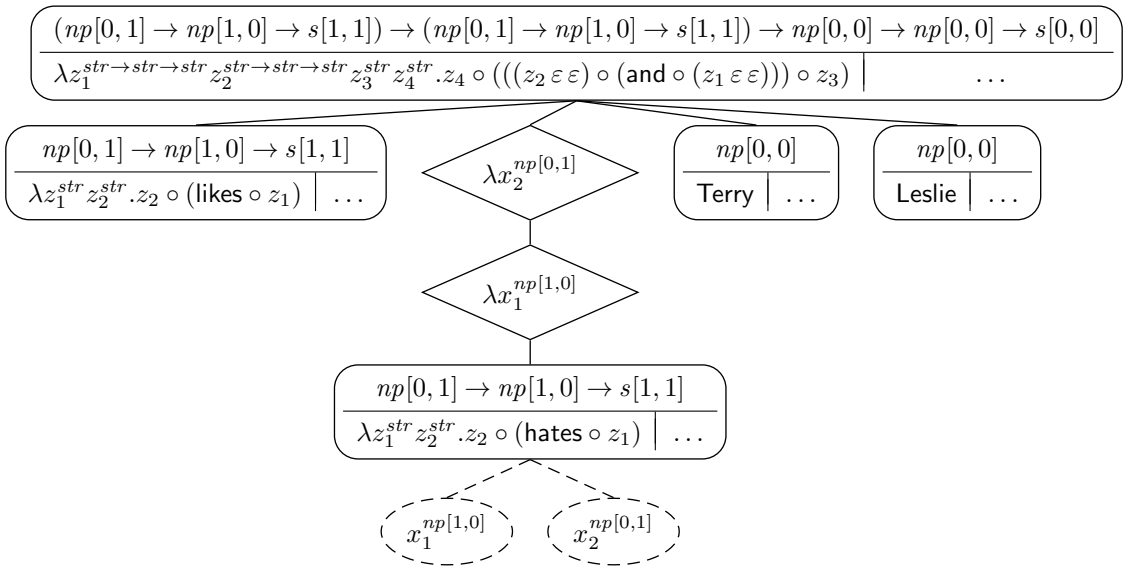


Figure 18: Failed derivation of Leslie likes and hates Terry with the meaning "Leslie likes Terry and Terry hates Leslie".

$$
\begin{aligned}
[\textsc{lgap}\,0] \circ [\textsc{lgap}\,0] &\to [\textsc{lgap}\,0],\\
[\textsc{lgap}\,1] \circ [\textsc{lgap}\,0] &\to [\textsc{lgap}\,1],\\
\mathbf{O}^L\,[\textsc{lgap}\,1] &\to [\textsc{lgap}\,0],\\
\mathbf{O}^R\,[\textsc{lgap}\,l] &\to [\textsc{lgap}\,l] \quad \text{for } l \in \{0,1\}.
\end{aligned}
$$

The tree automaton for the RGAP feature is similarly extended with transitions for ε^L and $\mathbf{O}^L$. These two automata independently determine how each feature is passed around in the syntactic types of the ACG entries. Writing $[l,r]$ for $[\textsc{lgap}\,l, \textsc{rgap}\,r]$, we get the final ACG in Figure 17. For example, this grammar blocks the derivation in Figure 18 because of feature mismatches between the entry for hates and its two arguments.

A further complication is necessary to deal with entries that introduce "binders" of two or more right-peripheral gaps (or left-peripheral gaps), like the following entry for and conjoining two ditransitive verbs:

$$((((np\backslash s)/np)/np)\backslash(((np\backslash s)/np)/np))/(((np\backslash s)/np)/np) : \mathsf{and}$$

This requires distinguishing two kinds of right-peripheral gaps, $\varepsilon_1^R, \varepsilon_2^R$, and the corresponding markers of scope, $\mathbf{O}_1^R, \mathbf{O}_2^R$. The full translation to ACGs with markers and the

associated tree automata are described in the appendix (Section A).

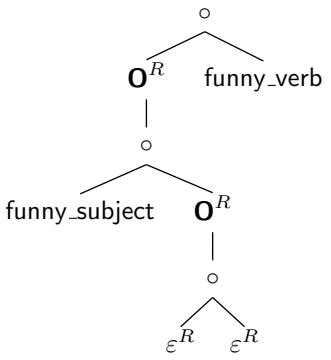
The ACG obtained by this method from an arbitrary Lambek grammar often undergenerates relative to the input Lambek grammar, since it does not allow nested uses of the / (or \) introduction rule. The following is the simplest kind of example that gives rise to this phenomenon:

$$s/(s/np) : \mathsf{funny\_subject}$$
$$(s/(s/np))\backslash s : \mathsf{funny\_verb}$$

The "η-expanded" form of the Lambek derivation for the sentence funny_subject funny_verb looks as follows:[25]

$$\dfrac{\mathsf{funny\_subject} \quad \dfrac{\dfrac{[s/np]^2 \quad [np]^1}{s}/E}{s/np}/I,1}{s/(s/np)} \quad \dfrac{\dfrac{s}{s/(s/np)}/I,2 \qquad \mathsf{funny\_verb}}{s}\backslash E \qquad (s/(s/np))\backslash s$$

The corresponding derivation of the ACG with markers gives the tree



which is not accepted by the tree automaton regulating the positions of ε^R. However, a situation like this does not seem to arise in practice in Lambek grammars for natural language, where two right-peripheral gaps "bound" by distinct binders (in this case $s/(s/np)$ and $(s/(s/np))\backslash s$) occur unbound in the same sub-derivation (in this case the derivation ending in the topmost occurrence of s).

Note that an analogous case involving both kinds of slashes poses no problem:

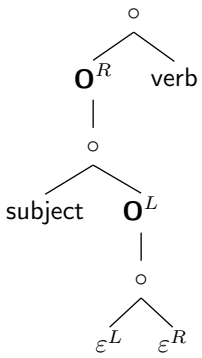
$$s/(np\backslash s) : \mathsf{subject}$$
$$(s/(np\backslash s))\backslash s : \mathsf{verb}$$

The Lambek derivation (in η-expanded form) for subject verb is

$$\dfrac{\mathsf{subject} \quad \dfrac{\dfrac{[np]^1 \quad [np\backslash s]^2}{s}\backslash E}{np\backslash s}\backslash I,1}{s/(np\backslash s)} \quad \dfrac{\dfrac{s}{s/(np\backslash s)}/I,2 \qquad \mathsf{verb}}{s}\backslash E \qquad (s/(np\backslash s))\backslash s$$

[25] I'm displaying a Lambek derivation in η-expanded form because my method of translation effectively works on the η-expanded form of the one-line derivation consisting of the type assigned to a terminal. Of course, there is no need to restrict our attention to such derivations when discussing a Lambek grammar.

The corresponding ACG derivation will give the tree



which is accepted by the two automata (for ε^R and for ε^L).

Another hypothetical situation where the proposed translation will fail for the same reason is when a Lambek grammar has entries like

$$\begin{aligned}(np\backslash s)/(vp\_inf/np) &: \mathsf{is\_hard}\\ vp\_inf/(np\backslash s) &: \mathsf{to}\\ ((np\backslash s)/np)/np &: \mathsf{teach}\\ s\_int/(s/np) &: \mathsf{who}, \mathsf{which\_language}\\ (np\backslash s)/s\_int &: \mathsf{wonders}\\ np &: \mathsf{Leslie}, \mathsf{Robin}, \mathsf{Haskell}\end{aligned}$$

This grammar generates Leslie wonders who$_1$ Haskell$_2$ is_hard to teach t_1 t_2 (with crossed dependencies), but not Leslie wonders which_language$_2$ Robin$_1$ is_hard to teach t_1 t_2 (with nested dependencies) (Figure 19). (The ACG translation rejects both sentences.) Of course, a Lambek grammar like this does not even remotely resemble a correct description of a fragment of English; neither *wh*-movement nor *tough*-movement is restricted to right periphery, and if anything, the word order exhibiting nested dependencies often seems to be the preferred one.[26] In general, it seems to me that this peculiar ability of Lambek's directional slashes to enforce crossed, as opposed to nested, dependencies between binders and gaps is never utilized in descriptions of natural language.

The present automatic method of translating a Lambek grammar into an ACG seems to me to provide a good enough approximation of the input Lambek grammar for the purpose of linguistic description. I am not, however, advocating to build an ACG-based linguistic theory on the basis of this translation. When you are dealing with an expression whose meaning is a function, there is always a choice between analyzing its form as a λ-abstract or as a simple string. If you choose the latter, the syntactic type of the expression must be atomic. If you choose the former, there is a further choice between giving the expression a functional syntactic type or an atomic syntactic type. In the largely CFG-based analysis of Sections 2.1, 2.3, 3, and 4.1, all standard constituents except that, whether, and and were analyzed as simple strings in the form dimension and given atomic syntactic types, while non-standard constituents and constituents with gaps were analyzed as functions in the form dimension and given functional syntactic types.[27] Alternatively, you could give non-standard constituents atomic types while analyzing them as functions in the form

[26] Presumably, the particular sentences at hand are both ungrammatical in English because they involve extraction of the first object in a double object (dative shift) construction.

[27] Of course, it is also easy to treat that, whether, and and like all other standard constituents.

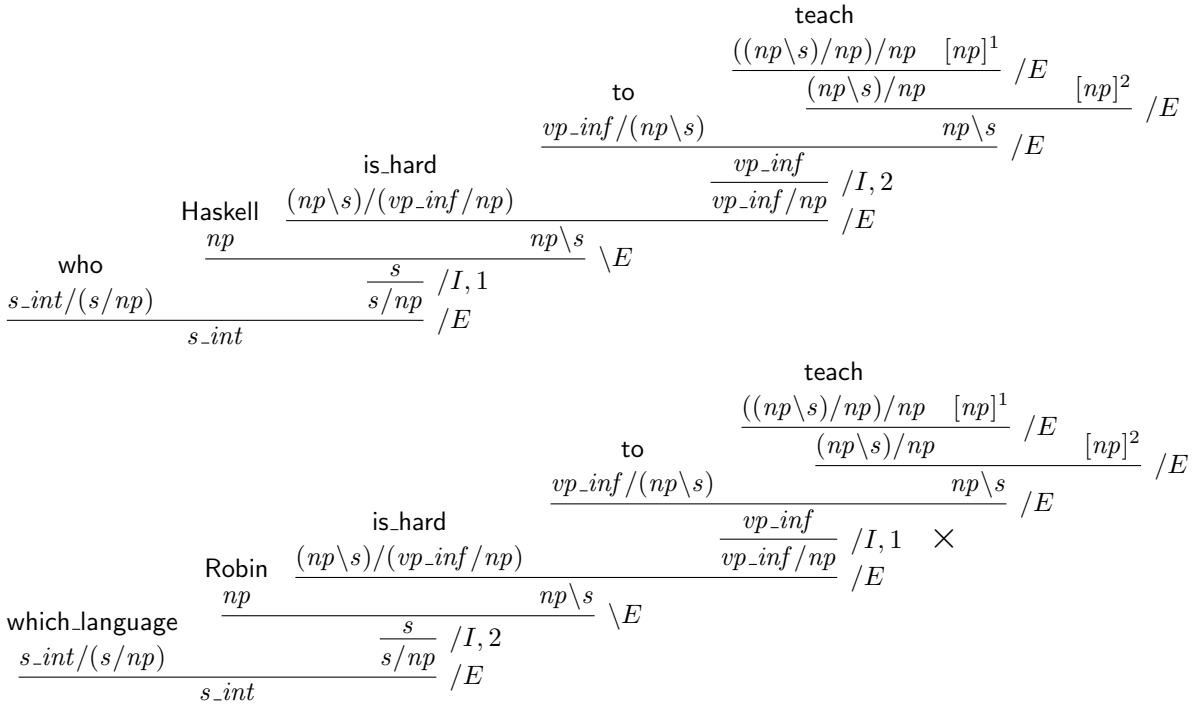


Figure 19: A Lambek derivation with crossed dependencies (above) and a failed derivation with nested dependencies (below).

dimension, on the model of the TAG-style treatment of *wh*-extraction in Section 2.2. Or you could analyze some standard constituents as functions in the form dimension and give them functional types, as in the translation of Lambek grammars. These choices (strings vs. functions, atomic vs. functional syntactic types) need not be made uniformly; the decision can be made on a case-by-case basis, taking into account properties of a particular construction at hand. Even when you decide to adopt a Lambek-style analysis of a certain construction, the output of the automatic translation should be regarded as a rough prototype which you can try to refine "manually" to suit the particular properties of the construction in question. The output of my translation method applied to an existing Lambek grammar (or a "hybrid" extension thereof, see below) should be viewed as a kind of baseline—the *worst* you could do in a linguistic theory based on the ACG formalism.

4.3 Gapping

The method of Section 4.2 is also applicable to Kubota and Levine's (2014a; 2014b) formalism of hybrid type-logical grammars, which is a kind of amalgam of Lambek grammars and ACGs. Hybrid type-logical grammars use both the directional slashes of the Lambek calculus and the "non-directional" implication from the simply typed lambda calculus. A *hybrid type* is either a directional type of the Lambek calculus or of the form $A \to B$, where A and B are hybrid types. For example, Kubota and Levine (2014b) postulated the following entry for the conjunction and in the Gapping construction, where $tv = (np\backslash s)/np$:[28]

$$\begin{aligned}&(\,(tv \to s) \to (tv \to s) \to tv \to s,\ \lambda z_1^{str\to str} z_2^{str\to str} z_3^{str}.(z_2 z_3) \circ (\mathsf{and} \circ (z_1\,\varepsilon)),\\ &\qquad \lambda z_1^{(e\to e\to t)\to t} z_2^{(e\to e\to t)\to t} z_3^{e\to e\to t}.\ \wedge^{t\to t\to t}\ (z_1(\lambda y^e x^e.z_3 y x))\,(z_2(\lambda y^e x^e.z_3 y x))\,).\end{aligned}$$

[28]This is actually an instance of a more general schema where, instead of tv, one has any Lambek type of the form $Y_0\backslash s/Y_1/\ldots/Y_n$ or $s/Y_0/Y_1/\ldots/Y_n$ with $n \geq 1$.

This entry together with the entries of the Lambek grammar in (15) license a hybrid derivation of the form-meaning pair

$$\begin{aligned}&(\,(\mathsf{Leslie} \circ (\mathsf{likes} \circ \mathsf{Robin})) \circ (\mathsf{and} \circ (\mathsf{Robin} \circ (\varepsilon \circ \mathsf{Terry}))),\\ &\qquad\qquad \wedge^{t\to t\to t}\ (\mathbf{likes\ Robin\ Leslie})\,(\mathbf{likes\ Terry\ Robin})\,).\end{aligned}$$

A naive translation of this fragment into an ACG, with the following entry for and, would make the sentence Leslie likes Robin and Robin Terry ambiguous between the actual reading "Leslie likes Robin and Robin likes Terry" and the reading "Robin likes Leslie and Terry likes Robin":[29]

$$\begin{aligned}&(\,((np \to np \to s) \to s) \to ((np \to np \to s) \to s) \to (np \to np \to s) \to s,\\ &\quad \lambda z_1^{(str\to str\to str)\to str}\, z_2^{(str\to str\to str)\to str}\, z_3^{str\to str\to str}.\\ &\qquad (z_2(\lambda y^{str} x^{str}.x \circ ((z_3\,\varepsilon\,\varepsilon) \circ y))) \circ (\mathsf{and} \circ (z_1(\lambda y^{str} x^{str}.x \circ (\varepsilon \circ y)))),\\ &\quad \lambda z_1^{(e\to e\to t)\to t}\, z_2^{(e\to e\to t)\to t}\, z_3^{e\to e\to t}.\ \wedge^{t\to t\to t}\ (z_1(\lambda y^e x^e.z_3 yx))\,(z_2(\lambda y^e x^e.z_3 yx))\,).\end{aligned}$$

Again, we can use the LGAP and RGAP features to deal with this problem, without abandoning the style of analysis employed by Kubota and Levine (2014b):[30]

$$\begin{aligned}&(\ ((np[0,r_1] \to np[l_1,0] \to s[l_1,r_1]) \to s[0,r]) \to\\ &\quad ((np[0,r_2] \to np[l_2,0] \to s[l_2,r_2]) \to s[l,0]) \to (np[0,1] \to np[1,0] \to s[1,1]) \to s[l,r],\\ &\quad \lambda z_1^{(str\to str\to str)\to str}\, z_2^{(str\to str\to str)\to str}\, z_3^{str\to str\to str}.\\ &\qquad (z_2(\lambda y^{str} x^{str}.x \circ (\mathbf{O}^R\,(\mathbf{O}^L\,(z_3\,\varepsilon^R\,\varepsilon^L)) \circ y))) \circ (\mathsf{and} \circ (z_1(\lambda y^{str} x^{str}.x \circ (\varepsilon \circ y)))),\\ &\qquad \lambda z_1^{(e\to e\to t)\to t}\, z_2^{(e\to e\to t)\to t}\, z_3^{e\to e\to t}.\wedge^{t\to t\to t}\,(z_1(\lambda y^e x^e.z_3 yx))\,(z_2(\lambda y^e x^e.z_3 yx))\,).\end{aligned}$$

The way overgeneration is blocked here (Figure 20) is entirely analogous to the case of transitive verb conjunction (Figure 18).

4.4 A Closer Look at Gapping

Kubota and Levine's (2014b) decision to exclusively use Lambek types like $tv = (np\backslash s)/np$ as the possible categories of the gap (the elided material in the second conjunct of Gapping) is puzzling. It has been widely recognized that the gap can be discontiguous:

(16) Max seemed to be trying to force Ted to leave the room, and Walt [~~seemed to be trying to force~~] Ira [~~to leave the room~~] (Jackendoff, 1971, p. 25)

(17) Arizona elected Goldwater Senator, and Pennsylvania [~~elected~~] Schweiker [~~Senator~~] (Jackendoff, 1971, p. 24)

(18) Jack begged Elsie to get married, and Wilfred [~~begged~~] Phoebe [~~to get married~~] (Jackendoff, 1971, p. 24)

(19) Max wanted Ted to persuade Alex to get lost and [~~Max wanted~~] Walt [~~to persuade~~] Ira [~~to get lost~~] (Hankamer, 1973, pp. 26–27)

(20) John took Harry to the movies, and Bill [~~took~~] Mike [~~to the movies~~] (Sag, 1976, p. 218)

[29] There is an alternative translation considered by Moot (2014) and Kubota and Levine (2014b) which gives the sentence the reading "Leslie likes Robin and Terry likes Robin". This entry is shown in Figure 21.

[30] I'm showing the grammar with markers for the convenience of the reader; the markers are not actually present in the output ACG of the translation.

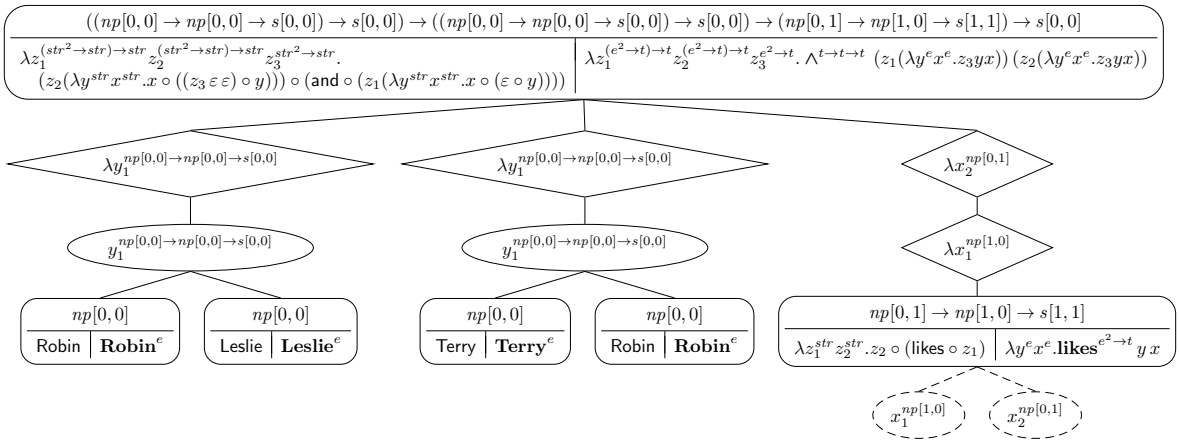


Figure 20: Failed derivation of Leslie likes Robin and Robin Terry with the meaning "Robin likes Leslie and Terry likes Robin".

(21) John persuaded Dr. Thomas to examine Mary, and Bill [~~persuaded~~] Dr. Jones [~~to examine Mary~~] (Sag, 1976, p. 225)

(22) Joe covered the floor with red paint, and Alice [~~covered~~] the walls [~~with red paint~~] (Neijt, 1980, p. 79)

(23) Joe painted his boat red, and Alice [~~painted~~] her car [~~red~~] (Neijt, 1980, p. 79)

(24) Some people want all doors to open to the left and others [~~want~~] all windows [~~to open to the left~~] (Neijt, 1980, p. 160)

In all of these examples, there is elided material to the right of the second remnant, so it seems that a hybrid type-logical grammar would need to assign the gap either a hybrid type $np \to (np \backslash s)$ or a simple type $np \to np \to s$.[31]

Examples like the following seem to require $np \to np \to s$ as the category of the gap:

(25) Max ordered Ted to persuade Alex to get lost and [~~Max ordered~~] Walt [~~to persuade~~] Ira [~~to get lost~~]

(26) I asked Peter to take Susan home, and [~~I asked~~] John [~~to take~~] Wendy [~~home~~]

(27) Rarely does John call Mary at home, and [~~rarely does~~] Mary [~~call~~] John [~~at home~~]

This puts hybrid type-logical grammars in the same quandary that plagued the naive ACG translation of a Lambek grammar (Section 4.3). Assuming hybrid entries in Figure 21, the sentence I asked Peter to take Susan home and John Wendy would come out as ambiguous between the reading indicated in (26) and the reading "I asked Peter to take Susan home, and I asked Wendy to take John home" (among other readings).

Sentences like (25) and (26) seem to be generally acceptable. If so, the true generalization about the word order in Gapping may be the following:

(28) In the first conjunct of Gapping, the correspondent of the first remnant must precede the correspondent of the second remnant.

It should be clear that (28) can be expressed as a regular constraint, if we mark the positions of the correspondents in the first conjunct of Gapping:

[31] In Hankamer's example (19), the hybrid type $np \to (np \backslash s)$ will work under the analysis where Gapping occurs in the infinitival clause [Ted to persuade Alex to get lost and Walt [~~to persuade~~] Ira [~~to get lost~~]].

np	: I	: $\mathbf{me}^{e}$
np	: Peter	: $\mathbf{Peter}^{e}$
np	: Susan	: $\mathbf{Susan}^{e}$
np	: John	: $\mathbf{John}^{e}$
np	: Wendy	: $\mathbf{Wendy}^{e}$
$vp\_inf/(np\backslash s)$	: to	: $\lambda y^{e\to t} x^{e}.yx$
$(np\backslash s)/vp\_inf/np$	: asked	: $\lambda y^{e} z^{e\to t} x^{e}.\mathbf{ask}^{e\to(e\to t)\to e\to t}\, y\,(zy)\,x$
$(np\backslash s)/pp/np$	: take	: $\lambda y^{e} z^{e} x^{e}.\mathbf{take}^{e\to e\to t}\, z\, y\, x$
pp	: home	: $\mathbf{home}^{e}$
$((np^2 \to s) \to s) \to$ $((np^2 \to s) \to s) \to$ $(np^2 \to s) \to s$	: $\lambda z_1^{(str^2\to str)\to str}$ $z_2^{(str^2\to str)\to str} z_3^{str^2\to str}.$ $(z_2(\lambda y^{str} x^{str}.z_3 yx)) \circ$ $(\mathsf{and} \circ (z_1(\lambda y^{str} x^{str}.x \circ (\varepsilon \circ y))))$	: $\lambda z_1^{(e^2\to t)\to t} z_2^{(e^2\to t)\to t} z_3^{e^2\to t}.$ $\wedge^{t\to t\to t}\,(z_1(\lambda y^{e} x^{e}.z_3 yx))$ $(z_2(\lambda y^{e} x^{e}.z_3 yx))$

Figure 21: A hybrid type-logical grammar for discontiguous Gapping.

$$\lambda z_1^{(str\to str\to str)\to str} z_2^{(str\to str\to str)\to str} z_3^{str\to str\to str}.$$
$$(z_2(\lambda y^{str} x^{str}.\,\mathbf{O}^{<}\,(z_3(\mathbf{C}_2\,y)(\mathbf{C}_1\,x)))) \circ (\mathsf{and} \circ (z_1(\lambda y^{str} x^{str}.x \circ (\varepsilon \circ y)))). \quad (29)$$

One can then capture this regular constraint with a syntactic feature, again using the general method of Kanazawa (2006). The resulting entry would work for all cases of Gapping in which the correspondents/remnants are two noun phrases. Note that the method is applicable no matter what regular constraint governs the positions of the correspondents/remnants. If, as has often been argued (Kuno, 1976; Neijt, 1980; Coppock, 2001; Johnson, 2014), the relative positions of the correspondents/remnants obey some (but perhaps not all) of the island constraints governing *wh*-extraction, those constraints can also be captured by the syntactic feature, as long as they are regular.[32]

Actually, even the status of the generalization (28) may be open to question. Sag et al. (1985) used the following example to question Hudson's (1982) claim that in Gapping "the order of constituents in the second conjunct ... parallel the order of the corresponding constituents in the first conjunct" (Hudson, 1982, p. 548):

(30) A policeman walked in at 11, and at 12, a fireman (Sag et al., 1985, p. 158)

The next examples are from Hankamer (1979, pp. 151–152):

(31) a. The beans, Harry cooked, and the potatoes, Henry
b. The beans, Harry cooked, and Henry, the potatoes

The following word order is also perfectly acceptable, given the right context:[33]

(32) Harry cooked the beans, and the potatoes, Henry

It seems likely that the reverse word order in (30), (31b), and (32) is the result of interaction between Gapping and a separate process of Topicalization. Hankamer (1979),

[32]In addition to constraints that hold of *wh*-extraction, it has been argued that Gapping obeys the Tensed S Condition (Neijt, 1980).

[33]For example, (32) is natural in the following dialogue:

(i) A. Gee, the beans and the potatoes are good! Did Harry cook them again?
B. No. Today, Harry cooked the beans, and the potatoes, Henry.

based on (31a) and (31b), concluded that the transformation of Topicalization follows Gapping. In terms of an analysis along the line of Kubota and Levine (2014b), this means that the first and the second arguments of the Gapping entry for and can independently undergo Topicalization.[34] If so, the implementation suggested above of the generalization (28) in terms of a syntactic feature can be maintained. Whether or not this is on the right track, data like (30), (31a), (31b), and (32) show that one needs an account of the effect of Topicalization on possible intonations and topic-focus structures of utterances in order to tell whether a given analysis of Gapping (and Topicalization) generates ungrammatical form-meaning pairs.[35]

To end this inconclusive discussion, no matter what constraint the grammar should ultimately impose on the order of the correspondents/remnants in Gapping, it seems fairly clear that Lambek's directional slashes are not the right tool for this purpose.

5 Conclusion

I would like to end by answering some questions that might be raised against this work.

What about the Pentus construction?

> *Pentus (1993; 1997) showed that a Lambek grammar can be translated into an equivalent context-free grammar, and Kanazawa and Salvati (2013) showed that the construction preserves the string-meaning relation. By de Groote's (2001) encoding, this means that every Lambek grammar has an equivalent ACG. Why do you need another translation which is only an approximation?*

Pentus's construction drastically changes the derivations of the input Lambek grammar. The ACG obtained this way is a particularly simple kind of second-order ACG and does not use λ-abstraction in the derivation. Along with Moot (2014) and Kubota and Levine (2014a,b), I am interested in the possibility of translating Lambek grammars into ACGs in such a way that structures of derivations are preserved.[36]

[34] In ACGs/hybrid type-logical grammars, Topicalization can be handled in a similar way to *wh*-extraction.

[35] Winkler (2005, p. 192) proposed the following principle:

> Contrastive Topic and Focus Principle:
> In gapping, the first remnant is a contrastive topic, the second remnant a contrastive focus.
> The gapped elements must be given.

This does not always seem to be the case, however:

(i) (Did Gwendolyn play poker and Alan canasta?)
No, Alan played poker, and Gwendolyn canasta. (Sag, 1976, p. 192)

[36] This is not to say that Pentus's conversion of Lambek grammars to CFGs is necessarily without linguistic interest. Moot (2014, p. 59) stresses that Pentus's construction brings about an exponential blowup in the size of the grammar. While this is literally true of the CFG defined by Pentus (1997, Theorem 2), this CFG contains a lot of useless nonterminals and superfluous rules, and should not be the focus of our attention when we are interested in grammar size. We can obtain a much more compact CFG by adhering to Pentus's proof closely (see Pentus, 1997, Lemma 7) and eliminating some obvious redundancies. It is also worth pointing out that even if it turns out that Pentus's construction must lead to an exponentially larger grammar in the worst case, this will not mean that exponential blowup is inevitable. A particular conversion method can only establish an upper bound on the increase in size when going from one formalism to another; establishing that such an upper bound is the best possible one would require an entirely different kind of proof. Note that the NP-completeness of derivability in the product-free Lambek calculus (Savateev, 2012) implies that any method of converting Lambek grammars to CFGs must have non-polynomial time complexity, but says nothing about the size of the output.

What are syntactic features?

> *Doesn't addition of syntactic features require extension of the type theory behind ACGs?*

I prefer to think of features as abbreviatory devices outside of the formalism of ACGs (see the clarification at the end of Section 1). An entry with variables as feature values abbreviates all of its instantiations. Since I only consider finite-valued features, this means that I do not go beyond the expressive power of ACGs as originally defined by de Groote (2001). An alternative is to embrace features as part of the formalism, using a type-theoretic extension of ACGs (de Groote and Maarek, 2007). In the particular case of finite-valued features, I see little advantage in doing so.

How many features are needed?

> *You have used one feature to express island constraints on wh-extraction, one feature to handle right-peripheral extraction in Right Node Raising, another feature to mimic Lambek's backslash, and yet another to constrain the relative order of the correspondents in the first conjunct of Gapping. If you need a new feature every time you want to express a regular constraint, don't you end up with an extremely large number of features?*

Beside these examples, I do not expect an ACG-based linguistic theory to need many more features like them. I consider the last three features to be natural ones for constraining the positions of the gaps (right- and left-peripherality and relative order between two gaps[37]). Note that these features are capable of capturing any Boolean combination of the relevant regular constraints; if, for instance, it turns out to be desirable to restrict the gap in a certain construction to a *non-right-peripheral* position, one can easily do so without introducing a new feature.[38]

Doesn't adding syntactic features result in an undesirable increase in the size of the grammar?

> *If you use variables as feature values to abbreviate grammar entries, the size of the grammar will be very large compared to the abbreviated notation. Isn't that a problem?*

I assume that a grammar is represented in the linguistic theory (or in the brain) in compressed form, using features and a host of other abbreviatory devices. The size of the uncompressed grammar may become an issue if it is assumed that the compressed representation needs to be fully decompressed in order to be put to use. This is a rather implausible assumption. Even if an individual entry needs to be "multiplied out" with concrete feature values in order to be used, it is reasonable to assume that only a limited number of entries are activated on any single occasion of language use (like sentence comprehension or production). It is hard to be concrete without a good model of parsing and generation (which we only have for second-order ACGs (Kanazawa, 2007, 2011)), but in terms of de Groote's (2015) parsing schema for higher-order ACGs, it is a trivial exercise to accommodate features like the ones we've been discussing, without using any decompression.

[37] In the marked ACG entry (29), the markers $\mathbf{C}_1$ and $\mathbf{C}_2$ effectively mark the positions of the gaps in the third argument of the entry.

[38] Depending on the style of analysis, you may not even need all of these features; if you adopt a CFG-based style of analysis of English, you may not ever need the LGAP feature, since, for example, the atomic type vp takes the place of $np\backslash s$.

Don't syntactic features increase the computational complexity of the linguistic theory?

> *Granted that the decompressed grammar need not be explicitly stored even temporarily, doesn't the presence of features still increase the computational complexity of parsing, relative to the complexity of parsing with ACGs given in uncompressed form?*

This question is a little hard to make sense of, since the time complexity of parsing with the decompressed grammar certainly places an upper bound on the time complexity of parsing with the compressed grammar, modulo the overhead incurred by decompression. The suggestion is that the dependence of the time complexity of parsing on the size of the grammar will be significantly higher in the presence of features than in their absence. In general, allowing an arbitrary number of finite-valued features changes a tractable parsing problem into an intractable one (Barton et al., 1987, Chapter 3), but here we are not dealing with a general case. What we have are a small number of features that express deterministic bottom-up finite tree automata. It is not hard to see that such features do not increase the degree of global ambiguity; the fact that they express deterministic bottom-up finite tree automata means that any derivation tree that is legitimate except for possible feature mismatches allows at most one assignment of feature values. How much *local ambiguity* increases, which is the real cause of increased time complexity, depends on the parsing algorithm. Again, since we do not have a good model of parsing for ACGs in general, it is hard to be concrete, but it is likely that determinism will help in reducing local ambiguity as well. In terms of second-order ACGs, whose computational complexity properties are well understood, the universal recognition problem is already PSPACE-hard, and one needs to place a bound on the size of $\sigma(\alpha)$ (for syntactic types α of grammar entries) to make it tractable (Kanazawa, 2011).[39] When the size of $\sigma(\alpha)$ and the number of features are both bounded, even explicitly decompressing the grammar only leads to a polynomial increase in grammar size, so the problem remains tractable in the presence of features.

Why not go GPSG and let the feature mechanism handle extraction itself?

> *The way the* GAP *feature behaves is similar to the SLASH feature in GPSG (Gazdar et al., 1985). If such a feature is needed, why don't you just use the SLASH feature instead of λ-abstraction? Likewise, left- and right-peripheral extraction may be handled by LSLASH and RSLASH features.*

I don't have a good answer to this question. Some people think λ-abstraction in derivations is problematic, and this feature of ACGs should be abandoned (Kiselyov, 2015). One point in favor of λ-abstraction in derivations may be that it allows an ACG to be written in a style very close to common practices of linguists. For example, it seems straightforward to express in ACGs not only Montague's (1973) fragment but also much of what's found in Heim and Kratzer's (1998) textbook, using the general style of CFG plus extraction. Features can be conveniently left aside when they are not important.

Why not go hybrid instead of adding syntactic features?

> *Even if ACGs can simulate Lambek's directional slashes with syntactic features, isn't using Lambek's slashes simpler and more elegant? Why are you trying to replace them with crummy features?*

[39] Compare a similar bound that Barton et al. (1987, Chapter 9) placed on the length of rules in R-GPSG.

I consider ACGs to possess a kind of a priori appeal that hybrid type-logical grammars lack. The syntactic features considered in this paper are the most direct expressions of the relevant constraints on the positions of gaps. In contrast, there are natural constraints on the positions of gaps that cannot be captured by Lambek's slashes (see Section 4.4), and I hinted in Section 4.2 that the power of Lambek's slashes is very much underutilized in descriptions of natural language. Lambek's slashes seem to me to be neither sufficient nor necessary.

Isn't the ACG formalism too powerful?

> *ACGs are capable of incorporating in the form of a syntactic feature an arbitrary stipulation that is expressible as a regular constraint. Doesn't that make the formalism of ACGs too powerful and devoid of explanatory power?*

That is exactly right, and the same can be said of any reasonable grammar formalism. The relevant property, closure under intersection with regular sets, is intimately tied to tabular parsing (Lang, 1994), and seems to be an essential property of any mathematically well-behaved grammar formalism. Any explanatory power that a linguistic theory may have must come from a combination of the grammar formalism employed by the theory—which is neutral with respect to different applications—and specifically *linguistic* principles espoused by the theory.

References

Abeillé, Anne and Owen Rambow, eds. 2000. *Tree Adjoining Grammars*. Stanford, California: CSLI Publications.

Barton, G. Edward, Robert Berwick, and Eric Sven Ristad. 1987. *Computational Complexity and Natural Language*. Cambridge, Mass.: MIT Press.

Chaves, Rui P. 2012. On the grammar of extraction and coordination. *Natural Language and Linguistic Theory* 30:465–512.

Chomsky, Noam. 1965. *Aspects of the Theory of Syntax*. Cambridge, Mass.: MIT Press.

Comon, Hubert, Max Dauchet, Rémi Gilleron, Florent Jacquemard, Denis Lugiez, Christof Löding, Sophie Tison, and Marc Tommasi. 2008. *Tree Automata Techniques and Applications*. Available at `http://tata.gforge.inria.fr`. November 18, 2008.

Coppock, Elizabeth. 2001. Gapping: In defense of deletion. In *CLS 37: The Main Session*, 133–148. Chicago: Chicago Linguistic Society.

de Groote, Philippe. 2001. Towards abstract categorial grammars. In *Association for Computational Linguistics, 39th Annual Meeting and 10th Conference of the European Chapter, Proceedings of the Conference*, 148–155.

de Groote, Philippe. 2002. Tree-adjoining grammars as abstract categorial grammars. In *Proceedings of the Sixth International Workshop on Tree Adjoining Grammar and Related Frameworks (TAG+6)*, 145–150. Universitá di Venezia.

de Groote, Philippe. 2015. Abstract categorial grammar as linear logic programming. In M. Kuhlmann, M. Kanazawa, and G. Kobele, eds., *Proceedings of the 14th Meeting on the Mathematics of Language*. The Association for Computational Linguistics.

de Groote, Philippe and Sarah Maarek. 2007. Type-theoretic extensions of abstract categorial grammars. In R. Muskens, ed., *Workshop on New Directions in Type-theoretic Grammars*, 19–30.

de Groote, Philippe and Sylvain Pogodalla. 2004. On the expressive power of abstract categorial grammars: Representing context-free formalisms. *Journal of Logic, Language and Information* 13:421–438.

Frank, Robert. 2002. *Phrase Structure Composition and Syntactic Dependencies*. Cambridge, Mass.: MIT Press.

Gazdar, Gerald, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag. 1985. *Generalized Phrase Structure Grammar*. Cambridge, Mass.: Harvard University Press.

Hankamer, Jorge. 1973. Unacceptable ambiguity. *Linguistic Inquiry* 4:17–68.

Hankamer, Jorge. 1979. *Deletion in Coordinate Structures*. Outstanding Dissertations in Linguistics. New York: Garland.

Heim, Irene and Angelika Kratzer. 1998. *Semantics in Generative Grammar*. Oxford: Blackwell.

Hofmeister, Philip and Ivan Sag. 2010. Cognitive constraints and island effects. *Language* 86:365–415.

Hudson, Richard. 1982. Incomplete conjuncts. *Linguistic Inquiry* 13:547–550.

Jackendoff, Ray S. 1971. Gapping and related rules. *Linguistic Inquiry* 2:21–35.

Jacobson, Pauline. 2014. *Compositional Semantics*. Oxford: Oxford University Press.

Johnson, Kyle. 2014. Gapping. Manuscript available at `http://people.umass.edu/kbj/homepage/Content/gapping.pdf`.

Joshi, Aravind K. and Yves Schabes. 1997. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, eds., *Handbook of Formal Languages*, vol. 3, 69–123. Berlin: Springer.

Kanazawa, Makoto. 2006. Abstract families of abstract categorial languages. *Electronic Notes in Theoretical Computer Science* 165:65–80. Proceedings of the 13th Workshop on Logic, Language, Information and Computation (WoLLIC 2006).

Kanazawa, Makoto. 2007. Parsing and generation as Datalog queries. In *Proceedings of the 45th Annual Meeting of the Association for Computational Linguistics*, 176–183. Prague, Czech Republic.

Kanazawa, Makoto. 2009. Second-order abstract categorial grammars. Lecture notes for a course at ESSLLI 2009, available at `http://research.nii.ac.jp/~kanazawa/publications/esslli2009_lectures.pdf`.

Kanazawa, Makoto. 2011. Parsing and generation as Datalog query evaluation. Manuscript under review, available at `http://research.nii.ac.jp/~kanazawa/pagadqe.pdf`.

Kanazawa, Makoto. 2014. Almost affine lambda terms. In A. Indrzejczak, J. Kaczmarek, and M. Zawidzki, eds., *Trends in Logic XIII*, 131–148. Łódź: Łódź University Press.

Kanazawa, Makoto and Sylvain Salvati. 2013. The string-meaning relations definable by Lambek grammars and context-free grammars. In G. Morrill and M.-J. Nederhof, eds., *Formal Grammar, FG 2012/2013*, vol. 8036 of *Lecture Notes in Computer Science*, 191–208. Berlin: Springer.

Kanazawa, Makoto and Ryo Yoshinaka. 2005. Lexicalization of second-order ACGs. NII Technical Report NII-2005-012E, National Institute of Informatics, Tokyo.

Kiselyov, Oleg. 2015. Applicative abstract categorial grammars. In M. Kanazawa, L. Moss, and V. de Paiva, eds., *Proceedings of the Third Workshop on Natural Language and Computer Science*.

Kracht, Marcus. 2003. *The Mathematics of Language*. Berlin: Mouton de Gruyter.

Kroch, Anthony S. 1987. Unbounded dependencies and subjacency in a tree adjoining grammar. In A. Manaster-Ramer, ed., *The Mathematics of Language*, 143–172. Philadelphia: John Benjamins.

Kubota, Yusuke and Robert Levine. 2014a. Against ellipsis: Arguments for the direct licensing of 'non-canonical' coordinations. To appear in *Linguistics and Philosophy*.

Kubota, Yusuke and Robert Levine. 2014b. Gapping as hypothetical reasoning. To appear in *Natural Language and Linguistic Theory*.

Kuno, Susumu. 1976. Gapping: A functional analysis. *Linguistic Inquiry* 7:300–318.

Lang, Bernard. 1994. Recognition can be harder than parsing. *Computational Intelligence* 10(4):486–494.

Lazić, Ranko and Sylvain Schmitz. 2014. Non-elementary complexities for branching VASS, MELL, and extensions. In *Proceedings of the Joint Meeting of the Twenty-Third EACSL Annual Conference on Computer Science Logic (CSL) and the Twenty-Ninth Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 61:1–61:10. New York, NY, USA: ACM. ISBN 978-1-4503-2886-9.

Montague, Richard. 1973. The proper treatment of quantification in ordinary English. In J. Hintikka, J. Moravcsik, and P. Suppes, eds., *Approaches to Natural Language: Proceedings of the 1970 Stanford Workshop on Grammar and Semantics*. Dordrecht: Reidel.

Moot, Richard. 2014. Hybrid type-logical grammars, first-order linear logic and the descriptive inadequacy of lambda grammars. `https://hal.archives-ouvertes.fr/hal-00996724`.

Muskens, Reinhard. 2003. Language, lambdas, and logic. In G.-J. M. Kruijff and R. T. Oehrle, eds., *Resource-Sensitivity, Binding and Anaphora*, 23–54. Dordrecht: Kluwer.

Neijt, Anneke. 1980. *Gapping*. Dordrecht: Foris Publications, second revised edn.

Pentus, Mati. 1993. Lambek grammars are context free. In *Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science*, 429–433.

Pentus, Mati. 1997. Product-free Lambek calculus and context-free grammars. *Journal of Symbolic Logic* 62:648–660.

Phillips, Colin. 2006. The real-time status of island phenomena. *Language* 82:795–823.

Phillips, Colin. 2013. Some arguments and nonarguments for reductionist accounts of syntactic phenomena. *Language and Cognitive Processes* 28:156–187.

Pogodalla, Sylvain. 2009. Advances in abstract categorial grammars: Language theory and linguistic modeling. ESSLLI 2009 lecture notes, part II. `https://hal.inria.fr/hal-00749297`.

Pogodalla, Sylvain and Florent Pompigne. 2012. Controlling extraction in abstract categorial grammars. In P. de Groote and M.-J. Nederhof, eds., *Formal Grammar 2010/2011*, vol. 7395 of *Lecture Notes in Computer Science*, 162–177. Berlin: Springer.

Ranta, Aarne. 2004. Grammatical framework: A type-theoretical grammar formalism. *Journal of Functional Programming* 14:145–189.

Sag, Ivan Andrew. 1976. *Deletion and Logical Form*. Ph.D. thesis, Massachusetts Institute of Technology.

Sag, Ivan A., Gerald Gazdar, Thomas Wasow, and Steven Weisler. 1985. Coordination and how to distinguish categories. *Natural Language and Linguistic Theories* 3:117–171.

Salvati, Sylvain. 2005. *Problèmes de filtrage et problèmes d'analyse pour les grammaires catégorielles abstraites*. Ph.D. thesis, l'Institut National Polytechnique de Lorraine.

Salvati, Sylvain. 2007. Encoding second order string ACG with deterministic tree walking transducers. In S. Wintner, ed., *Proceedings of FG 2006: The 11th conference on Formal Grammar*, FG Online Proceedings, 143–156. Stanford, CA: CSLI Publications.

Savateev, Yury. 2012. Product-free lambek calculus is NP-complete. *Annals of Pure and Applied Logic* 163:775–788.

Seki, Hiroyuki, Takashi Matsumura, Mamoru Fujii, and Tadao Kasami. 1991. On multiple context-free grammars. *Theoretical Computer Science* 88:191–229.

Vijay-Shanker, K., David J. Weir, and Aravind K. Joshi. 1987. Characterizing structural descriptions produced by various grammatical formalisms. In *25th Annual Meeting of the Association for Computational Linguistics*, 104–111.

Winkler, Susanne. 2005. *Ellipsis and Focus in Generative Grammar*, vol. 81 of *Studies in Generative Grammar*. Berlin: Mouton de Gruyter.

Worth, Chris. 2014. The phenogrammar of coordination. In *Proceedings of the EACL 2014 Workshop on Type Theory and Natural Language Semantics (TTNLS)*, 28–36. Association for Computational Linguistics.

Yoshinaka, Ryo. 2006. *Extensions and Restrictions of Abstract Categorial Grammars*. Ph.D. thesis, University of Tokyo.

A Formal Definitions

The set $\mathrm{Tp}_{\backslash,/}(\mathcal{P})$ of *directional types* over a set $\mathcal{P}$ of atomic types is the smallest superset of $\mathcal{P}$ such that $A, B \in \mathrm{Tp}_{\backslash,/}(\mathcal{P})$ implies $A\backslash B, B/A \in \mathrm{Tp}_{\backslash,/}(\mathcal{P})$. The set $\mathrm{Tp}_{\to}(\mathcal{P})$ of *simple types* over $\mathcal{P}$ is the smallest superset of $\mathcal{P}$ such that $\alpha, \beta \in \mathrm{Tp}_{\to}(\mathcal{P})$ implies $\alpha \to \beta \in \mathrm{Tp}_{\to}(\mathcal{P})$. To each $A \in \mathrm{Tp}_{\backslash,/}(\mathcal{P})$, we associate a simple type $\overline{A} \in \mathrm{Tp}_{\to}(\mathcal{P})$ by $\overline{p} = p$ ($p \in \mathcal{P}$), $\overline{A\backslash B} = \overline{A} \to \overline{B}$, $\overline{B/A} = \overline{A} \to \overline{B}$.

For reasons of space, I use a term notation for natural deduction in the Lambek calculus. Suppose that for each $A \in \mathrm{Tp}_{\backslash,/}(\mathcal{P})$, there is a countably infinite supply $x^A, y^A, z^A, \dots$ of variables of type A. We define the set of *Lambek terms* and their *frontier*, which is a string of variables, as follows ($\circ$ stands for concatenation):

- x^A is a Lambek term of type A and its frontier is $\mathbf{fr}(x^A) = x^A$.
- If M is a Lambek term of type $A\backslash B$ and N is a Lambek term of type A, then $\mathrm{app}_{\backslash}\, MN$ is a Lambek term of type B and its frontier is $\mathbf{fr}(\mathrm{app}_{\backslash}\, MN) = \mathbf{fr}(N) \circ \mathbf{fr}(M)$.
- If M is a Lambek term of type B/A and N is a Lambek term of type A, then $\mathrm{app}_{/}\, MN$ is a Lambek term of type B and its frontier is $\mathbf{fr}(\mathrm{app}_{/}\, MN) = \mathbf{fr}(M) \circ \mathbf{fr}(N)$.
- If M is a Lambek term of type B, $\mathbf{fr}(M) = x^A \circ \gamma$, and γ contains no occurrence of x^A, then $\lambda_{\backslash} x^A.M$ is a Lambek term of type $A\backslash B$ and its frontier is $\mathbf{fr}(\lambda_{\backslash} x^A.M) = \gamma$.
- If M is a Lambek term of type B, $\mathbf{fr}(M) = \gamma \circ x^A$, and γ contains no occurrence of x^A, then $\lambda_{/} x^A.M$ is a Lambek term of type B/A and its frontier is $\mathbf{fr}(\lambda_{/} x^A.M) = \gamma$.

We assume familiarity with *(simply typed) λ-terms* and *linear λ-terms*. A Lambek term of type B with free variables $x_1^{A_1}, \dots, x_n^{A_n}$ is mapped to a simply typed linear λ-term of type $\overline{B}$ with free variables $x_1^{\overline{A_1}}, \dots, x_n^{\overline{A_n}}$, as follows:

$$\begin{aligned}
\overline{x^A} &= x^{\overline{A}},\\
\overline{\mathrm{app}_{\backslash}\, MN} &= \overline{M}\,\overline{N},\\
\overline{\mathrm{app}_{/}\, MN} &= \overline{M}\,\overline{N},\\
\overline{\lambda_{\backslash} x^A.M} &= \lambda x^{\overline{A}}.\overline{M},\\
\overline{\lambda_{/} x^A.M} &= \lambda x^{\overline{A}}.\overline{M}.
\end{aligned}$$

A *Lambek grammar* (over a terminal alphabet Σ and a set Δ of typed "meaning constants", with distinguished type $s \in \mathcal{P}$) is a finite set of triples of the form (A, a, N) ("lexical entries"), where $A \in \mathrm{Tp}_{\backslash,/}(\mathcal{P})$, $a \in \Sigma$, and N is a closed (not necessarily linear) λ-term of type $\tau(\overline{A})$ (which may contain constants from Δ in addition to bound variables) representing the meaning of a, where $\tau\colon \mathcal{P} \to \mathrm{Tp}_{\to}(\{e, t, \dots\})$ is a type substitution such that $\tau(s) = t$. If P is a Lambek term of distinguished type s with $\mathbf{fr}(P) = x_1^{A_1} \dots x_n^{A_n}$, and for each $i = 1, \dots, n$, (A_i, a_i, N_i) is a lexical entry of the grammar, then the pair $(a_1 \dots a_n, \overline{P}[x_i^{\overline{A_i}} := N_i]_{i=1}^n)$ belongs to the *string-meaning relation* defined by the grammar.

An *ACG* (over a terminal alphabet Σ and a set Δ of typed meaning constants, with distinguished type $s \in \mathcal{P}$) is a finite set of triples of the form (α, M, N) ("grammar entries"), where $\alpha \in \mathrm{Tp}_{\to}(\mathcal{P})$, M is a closed (linear) λ-term of type $\sigma(\alpha)$ containing contants from $\Sigma \cup \{\circ, \varepsilon\}$, and N is a closed λ-term of type $\tau(\alpha)$ containing constants from Δ, where $\sigma\colon \mathcal{P} \to \mathrm{Tp}_{\to}(\{str\})$ and $\tau\colon \mathcal{P} \to \mathrm{Tp}_{\to}(\{e, t, \dots\})$ are type substitutions such

that $\sigma(s) = str$ and $\tau(s) = t$. Symbols in Σ are assumed to have type str, and $\circ$ (for concatenation) and ε (for empty string) have types $str \to str \to str$ and str, respectively. If Q is a pure (i.e., constant-free) linear λ-term of type s with free variables $x_1^{\alpha_1}, \ldots, x_n^{\alpha_n}$, and for each $i = 1, \ldots, n$, (α_i, M_i, N_i) is an entry of the grammar, then the pair

$$(Q[x_i^{\alpha_i} := M_i]_{i=1}^n,\, Q[x_i^{\alpha_i} := N_i]_{i=1}^n)$$

is in the string-meaning relation defined by the ACG.

The naive translation from Lambek grammars to ACGs maps a lexical entry (A, a, N) of a Lambek grammar into $(\overline{A}, \mathrm{acg}_A\, a, N)$. The combinators acg_A of type $str \to \widetilde{A}$, where $\widetilde{A} = \sigma(\overline{A})$, $\sigma(p) = str$ for all $p \in \mathcal{P}$, together with the accompanying combinators lamb_A of type $\widetilde{A} \to str$, are defined as follows:

$$\begin{aligned}
\mathrm{acg}_p\, x^{str} &= x, & \mathrm{lamb}_p\, x^{str} &= x,\\
\mathrm{acg}_{A\backslash B}\, x^{str} &= \lambda y^{\widetilde{A}}.\, \mathrm{acg}_B\, ((\mathrm{lamb}_A\, y) \circ x), & \mathrm{lamb}_{A\backslash B}\, x^{\widetilde{A}\to\widetilde{B}} &= \mathrm{lamb}_B\, (x\, (\mathrm{acg}_A\, \varepsilon)),\\
\mathrm{acg}_{B/A}\, x^{str} &= \lambda z^{\widetilde{A}}.\, \mathrm{acg}_B\, (x \circ (\mathrm{lamb}_A\, z)), & \mathrm{lamb}_{B/A}\, x^{\widetilde{A}\to\widetilde{B}} &= \mathrm{lamb}_B\, (x\, (\mathrm{acg}_A\, \varepsilon)).
\end{aligned}$$

It is easy to see that $\mathrm{lamb}_A\, (\mathrm{acg}_A\, x^{str}) = x$ holds for all A. Of course, $\mathrm{acg}_A(\mathrm{lamb}_A\, x^{\widetilde{A}}) = x$ holds only when A is atomic.

Lemma 1. *Let M be a Lambek term of type A in normal form with $\mathbf{fr}(M) = x_1^{A_1} \ldots x_n^{A_n}$.*

(i) *If M is not a $\lambda_\backslash$- or $\lambda_/$-abstract, then $\overline{M}[x_i^{\overline{A_i}} := \mathrm{acg}_{A_i}\, x_i]_{i=1}^n = \mathrm{acg}_A(\mathbf{fr}(M))$.*

(ii) $\mathrm{lamb}_A\, \overline{M}[x_i^{\overline{A_i}} := \mathrm{acg}_{A_i}\, x_i]_{i=1}^n = \mathbf{fr}(M)$.

Proposition 2. *If M is a Lambek term of an atomic type with $\mathbf{fr}(M) = x_1^{A_1} \ldots x_n^{A_n}$, then $\overline{M}[x_i^{\overline{A_i}} := \mathrm{acg}_{A_i}\, x_i]_{i=1}^n = \mathbf{fr}(M)$.*

This means that the output ACG of the naive translation generates all pairs generated by the input Lambek grammar. The problem is that it massively overgenerates.

Clearly, what's wrong in the naive translation is the two identical clauses

$$\mathrm{lamb}_{A\backslash B}\, x^{\widetilde{A}\to\widetilde{B}} = \mathrm{lamb}_B\, (x\, (\mathrm{acg}_A\, \varepsilon)), \quad \mathrm{lamb}_{B/A}\, x^{\widetilde{A}\to\widetilde{B}} = \mathrm{lamb}_B\, (x\, (\mathrm{acg}_A\, \varepsilon)).$$

In the former clause, the value of $x^{\widetilde{A}\to\widetilde{B}}$ should really be restricted to a function that places its argument at its left edge, so to speak, while in the latter, it should be restricted to a function that places its argument at its right edge. Worth's (2014) idea was to use subtyping to narrow down the domains of the combinators $\mathrm{lamb}_{A\backslash B}$ and $\mathrm{lamb}_{B/A}$. Here, we try to obtain a similar effect by refining the atomic types in $\overline{A} \to \overline{B}$, without going beyond the original architecture of the ACG.

We temporarily introduce new constant symbols $\varepsilon_1^L, \varepsilon_2^L, \ldots$ and $\varepsilon_1^R, \varepsilon_2^R, \ldots$ of type str and new function symbols $\mathbf{O}_1^L, \mathbf{O}_2^L, \ldots$ and $\mathbf{O}_1^R, \mathbf{O}_2^R, \ldots$ of type $str \to str$, and change the definitions of acg and lamb as follows:

$$\begin{aligned}
\mathrm{acg}'_p\, x^{str} &= x,\\
\mathrm{acg}'_{A\backslash B}\, x^{str} &= \lambda y^{\widetilde{A}}.\, \mathrm{acg}'_B((\mathrm{lamb}'_A\, 1\, 1\, y) \circ x),\\
\mathrm{acg}'_{B/A}\, x^{str} &= \lambda z^{\widetilde{A}}.\, \mathrm{acg}'_B\, (x \circ (\mathrm{lamb}'_A\, 1\, 1\, z)),\\
\mathrm{lamb}'_p\, n\, m\, x^{str} &= x,
\end{aligned}$$

$$\mathrm{lamb}'_{A\backslash B}\, n\, m\, x^{\widetilde{A}\to\widetilde{B}} = \mathbf{O}^L_n\left(\mathrm{lamb}'_B\,(n+1)\,m\,(x\,(\mathrm{acg}'_A\,\varepsilon^L_n))\right),$$
$$\mathrm{lamb}'_{B/A}\, n\, m\, x^{\widetilde{A}\to\widetilde{B}} = \mathbf{O}^R_m\left(\mathrm{lamb}'_B\,n\,(m+1)\,(x\,(\mathrm{acg}'_A\,\varepsilon^R_m))\right).$$

The combinator lamb'_A is of type $\mathit{int}\to\mathit{int}\to\widetilde{A}\to\mathit{str}$, where int is the type of positive integers. The integer arguments of lamb'_A are there to distinguish different left- (or right-) peripheral gaps bound by the same "binder". Note that we can recover $\mathrm{acg}_A\,a$ from $\mathrm{acg}'_A\,a$ by substituting ε for ε^L_i and ε^R_j and $\lambda x^{\mathit{str}}.x$ for $\mathbf{O}^L_i$ and $\mathbf{O}^R_j$.

The resulting ACG is then filtered through two finite tree automata. Given maximal values $n_{\max}$ and $m_{\max}$ for n and m (the two integer arguments of lamb'_A), the automata have $n_{\max}(n_{\max}+1)/2+1$ and $m_{\max}(m_{\max}+1)/2+1$ states, respectively. The states of the second automaton are $[\textsc{rgap}\,0]$ and $[\textsc{rgap}\,i\ldots j]$ with $1\le i\le j\le m_{\max}$, and its transitions are

$$\begin{aligned}
a &\to [\textsc{rgap}\,0] \quad \text{for each terminal } a,\\
\varepsilon^L_i &\to [\textsc{rgap}\,0] \quad \text{for } i=1,\ldots,n_{\max},\\
\varepsilon^R_i &\to [\textsc{rgap}\,i] \quad \text{for } i=1,\ldots,m_{\max},\\
[\textsc{rgap}\,0]\circ[\textsc{rgap}\,0] &\to [\textsc{rgap}\,0],\\
[\textsc{rgap}\,0]\circ[\textsc{rgap}\,1\ldots i] &\to [\textsc{rgap}\,1\ldots i] \quad \text{for } 1\le i\le m_{\max},\\
[\textsc{rgap}\,i\ldots j]\circ[\textsc{rgap}\,(j+1)\ldots k] &\to [\textsc{rgap}\,i\ldots k] \quad \text{for } 1\le i\le j<k\le m_{\max},\\
\mathbf{O}^L_i\,q &\to q \quad \text{for each state } q \text{ and } i=1,\ldots,n_{\max},\\
\mathbf{O}^R_1\,[\textsc{rgap}\,1] &\to [\textsc{rgap}\,0],\\
\mathbf{O}^R_j\,[\textsc{rgap}\,1\ldots j] &\to [\textsc{rgap}\,1\ldots(j-1)] \quad \text{for } 1<j\le m_{\max}.
\end{aligned}$$

The definition of the first automaton is completely symmetric. We use new atomic types of the form $p[\textsc{lgap}\,l,\textsc{rgap}\,r]$ as a feature-specified variant of p. Given an entry

$$(\alpha, M', N) = (\overline{A}, \mathrm{acg}'_A\,a, N)$$

obtained from a Lambek grammar entry (A,a,N) and a feature specified variant α' of α, the entry (α', M', N) is in the filtered grammar if for $i=1,2$, M' has type $(\alpha')_i$ under the typing τ'_i of the constants determined by the ith automaton, where $(\alpha')_i$ is defined inductively as follows:

$$\begin{aligned}
(p[\textsc{lgap}\,l,\textsc{rgap}\,r])_1 &= [\textsc{lgap}\,l],\\
(p[\textsc{lgap}\,l,\textsc{rgap}\,r])_2 &= [\textsc{rgap}\,r],\\
(\alpha'\to\beta')_i &= (\alpha')_i\to(\beta')_i.
\end{aligned}$$

The typing τ'_i is a simple kind of intersection typing that assigns a set of types to each constant. For example, $q_1\to q_2\to q\in\tau'_i(\circ)$ if and only if $q_1\circ q_2\to q$ is a transition of M_i.

If (α', M', N) is in the filtered grammar, with $M'=\mathrm{acg}'_A\,a$, then (α', M, N) is in the "final product" grammar, where $M=\mathrm{acg}_A\,a$.