Applicative Abstract Categorial Grammars in Full Swing

Oleg Kiselyov

Tohoku University, Japan
oleg@okmij.org

Abstract. Recently introduced Applicative Abstract Categorial Grammars (AACG) extend the Abstract Categorial Grammar (ACG) formalism to make it more suitable for semantic analyses while preserving all of its benefits for syntactic analyses. The surface form of a sentence, the abstract (tecto-) form, as well as the meaning are all uniformly represented in AACG as typed terms, or trees. The meaning of a sentence is obtained by applying to the abstract form a composition of elementary, deterministic but generally partial transformations. These term tree transformations are specified *precisely* and can be carried out mechanically. The rigor of AACG facilitates its straightforward implementation, in Coq, Haskell, Grammatical Framework, etc.
We put AACG through its paces, illustrating its expressive power and positive as well as negative predictions on the wide range of analyses: gender-marked pronouns, quantifier ambiguity, scoping islands and binding, crossover, topicalization, and inverse linking. Most of these analyses have not been attempted for ACG.
AACG offers a different perspective on linguistic side-effects, demonstrating compositionality not just of meanings but of transformations.

1 Introduction

One of the goals of the semantic analysis is comprehension: determining the meaning of an utterance. The subject of the analysis is usually not the raw utterance but rather more or less abstract form thereof, abstracting away peculiarities of pronunciation and writing and often declination, conjugation, etc. One may as well call this 'logical form'; we will stay however with 'abstract form'. Finding the right level of abstraction is one of the challenges.

One may discern two broad perspectives of the analysis. One is proof search: building a logical system derivation whose conclusion is directly related to the abstract form of an utterance. The successfully obtained derivation is taken as the evidence that the utterance is grammatical. The meaning is then read off the derivation. The proof search perspective is manifest in type-logical grammars; it also underlies general categorial grammars and Minimalist Grammars. Parsing as deduction is truly compelling and logically insightful. Yet it is often hard to characterize the space of possible derivations, to see that everything derivable will be judged by the native speakers as grammatical, and everything that judged as grammatical can be derived. In particular, it is difficult to get negative predictions: to prove that there really cannot be a derivation for a particular sentence, no matter how hard we try to build one.

Another perspective is most clearly described by Chung-chieh Shan in his papers on linguistic side effects [6]. (Discourse representation theory (DRT) is also in the same vein.) It is based not on proof but on computation: the source sentence is taken to be a program that computes the meaning as a logical formula. As any program, it may fail: raise an 'exception' or stop before computing any result. Thus in the computational perspective, the meaning is obtained as the result of a mostly deterministic, inherently partial, usually precisely specified and often mechanized process. The algorithmic nature of this process lets it lay a credible claim to 'real life', physiological relevance. On the downside, the computational process is rigid. It is also too easy to get bogged down to details (control operators, implementation of monads, etc.) Taken the computational perspective as the starting point, the present work aims to raise its level of abstraction, eliding implementation details and adding the 'right' amount of flexibility.

Applicative Abstract Categorial Grammars (AACG) recently introduced in [4] is a reformulation of Abstract Categorial Grammars (ACG) [1]. AACG supports syntactic analyses within the well-understood second-order ACG. In addition and in contrast to ACG, it lets us do semantic analyses without compromising syntactic ones. The meaning of a sentence is obtained from its abstract (parsed) form through a precisely specified process – so precise that it can be carried out mechanically, by a computer. The process is a composition of simple tree transformations. It may fail to deliver the meaning formula, thus predicting the unacceptability of the sentence.

AACG is formally introduced and compared to ACG in [4]. That paper however used rather simple illustrations. We now apply AACG to more interesting analyses, of phenomena exhibited in the following examples.

(1) Every girl$_i$'s father loves her$_i$ mother.
(1a) *Every girl$_i$'s father loves its$_i$ mother.
(1b) *Her$_i$ father loves every girl$_i$'s mother.
(1c) A girl$_i$ met every boy who liked her$_i$.
(2a) That John$_i$ left upset his$_i$ teacher.
(2b) *That every boy$_i$ left upset his$_i$ teacher.
(3a) Alice's present for him$_i$, every boy$_i$ saw.
(3b) *Every boy$_i$, his$_i$ mother likes.
(4a) At least two senators on every committee voted against the bill.
(4b) Two politicians spy on someone from every city.
(4c) Some man from every city secretly despises it.

We thus analyze, in §3, binding (1) with gender-marked pronouns (1a) and crossover (1b), scoping islands and binding (2), and cataphora and anaphora in topicalization (3). §4 deals with inverse linking (4). The examples have many subtleties: (1c) has in reality no reading with "every boy" outscoping "a girl". Likewise, in (4b), "two politicians" may scope either wider than "someone from every city" or narrower. There is however no reading in which "two politicians" scope between "someone" and "every city". The inadmissibility and the absence of the readings are reproduced in our analyses. These phenomena have not been dealt with, or even considered, within the original ACG. We drew most of the examples from the pioneering paper on linguistic side-effects [6], to contrast with its approach to direct compositionality. The inverse linking examples come from [5].

2 AACG Background and Quantifier Ambiguity

For reference we very briefly recall AACG, on the example of quantifier ambiguity; see [4] for all details.

In AACG, the surface form of a sentence, its various abstract (parsed) forms and the logical formula expressing the meaning – all are uniformly represented as typed terms (trees), in a T-language, Figure 1.

$$\begin{array}{llll} \text{Base types} & \upsilon & \text{T-Constants} & c \\ \text{T-Types} & \sigma ::= \upsilon \mid \sigma \rightarrowtail \sigma & \text{T-Terms} & d ::= c \mid d \bullet d \end{array}$$

$$\frac{}{c\colon \sigma} \qquad \frac{d_1\colon \sigma_1 \rightarrowtail \sigma_2 \quad d_2\colon \sigma_1}{d_1 \bullet d_2\colon \sigma_2}\,Tapp$$

Fig. 1. T-languages

T-types σ are formed from base types υ and the binary connective - $\rightarrowtail$ -. T-terms d are formed from constants c and the left-associative binary connective - $\bullet$ -. Each constant is assigned a T-type; the T-type of a composite term, if any, is determined by the inference rule (Tapp). A T-language is the set of all well-typed terms – or, a set of finite trees. On the latter view, the constants with their assigned types constitute a tree grammar. Different T-languages differ in their set of base types and their constants.

The following table shows three T-languages with the sample of constants, to be used throughout. T_S is the surface language of strings, with many string constants and one binary operation (written in infix) for string concatenation. T_A is for abstract forms: Curry's tecto-grammar. Its types are familiar categories. The silent constant cl combines an NP and a VP into a clause. These T-languages are first-order.

	υ	c
T_A	S, NP, N, VP	$John\colon NP$
		$man\colon N$
		$like\colon NP \rightarrowtail VP$
		$cl\colon NP \rightarrowtail VP \rightarrowtail S$
T_S	string	$\cdot$: string $\rightarrowtail$ string $\rightarrowtail$ string
		`"John"`, `"every"`, ... : string
T_L	e, t	conj, disj, ... : $t \rightarrowtail t \rightarrowtail t$
		john: e
		man: $e \rightarrowtail t$
		$\forall, \exists\colon (e \rightarrowtail t) \rightarrowtail t$
		$x, y, z, \ldots\colon e$
		$\hat{x}, \hat{y}, \hat{z}, \ldots\colon t \rightarrowtail (e \rightarrowtail t)$

The language T_L is to express meaning, as a logic formula. In addition to the standard logical constants it has an infinite supply of distinct *constants* $x, y, z, \ldots$ of the type e and the corresponding set of constants $\hat{x}$, etc., intended as binders. For example, the meaning of the sentence "a man left" is written in T_L as

$$\exists \centerdot (\hat{x} \centerdot (\mathsf{conj} \centerdot (\mathsf{man} \centerdot x) \centerdot (\mathsf{left} \centerdot x)))$$

We will often informally use the conventional logic notation however: $\exists x.\, \mathsf{man}\ x \wedge \mathsf{left}\ x$. Although T_L is sort of higher-order, it has no notion of reduction or substitution; its terms are just trees, but with bindings.

Determining the meaning of a sentence is transforming its abstract form T_A to the logical formula T_L. The transformation is easier to grasp if done in small steps. We will be using a variety of T_A languages to express the intermediate abstract forms. Each language adds to the core T_A described above new constants, whose set is summarized in the following table:

$$\begin{aligned} \mathit{every} &\colon N \rightarrowtail NP \\ a &\colon N \rightarrowtail NP \\ \mathit{var}_x, \mathit{var}_y, \ldots &\colon NP \\ U_x, U_y, \ldots &\colon N \rightarrowtail S \rightarrowtail S \\ E_x, E_y, \ldots &\colon N \rightarrowtail S \rightarrowtail S \\ \mathit{he}, \mathit{she}, \mathit{it} &\colon NP \end{aligned}$$

We will refer to the set $\mathit{var}_x, \mathit{var}_y, \ldots$ as just var, and similarly for U and E. These sets of constants are analogous to $x, y, \ldots, \hat{x}, \hat{y}, \ldots$ of the T_L language and represent (to be) bound variables and their binders. They are *distinct* from lambda-bound variables and are not subject to substitution, α-conversion or capture-avoidance. They are the variables and binders that Kobele [5] wished for and had to emulate in a complicated way.

2.1 Quantification and Quantifier Ambiguity

As an example, the sentence "every boy likes a girl" has the abstract form

$$\mathit{ex}_{bg} \overset{\text{def}}{=} \mathit{cl} \centerdot (\mathit{every} \centerdot \mathit{boy}) \centerdot (\mathit{like} \centerdot (a \centerdot \mathit{girl}))$$

written in the language $T_A \cup \{a, \mathit{every}\}$. One can easily imagine a transformation $\mathcal{L}_{syn}$ converting ex_{bg} to the T_S term

$$\texttt{"every"} \cdot \texttt{"boy"} \cdot \texttt{"like"} \cdot \texttt{"a"} \cdot \texttt{"girl"}$$

taken to be the surface form of the sentence. The transformation is a set of simple re-writing rules for converting T_A terms to T_S, as specified in Table 1. Alternatively, we can implement it as the mapping of

$\mathcal{L}_{syn}[boy]$	$\mapsto$	`"boy"`
$\mathcal{L}_{syn}[girl]$	$\mapsto$	`"girl"`
$\mathcal{L}_{syn}[a \centerdot d]$	$\mapsto$	$\texttt{"a"} \cdot \mathcal{L}_{syn}[d]$
$\mathcal{L}_{syn}[every \centerdot d]$	$\mapsto$	$\texttt{"every"} \cdot \mathcal{L}_{syn}[d]$
$\mathcal{L}_{syn}[like \centerdot d]$	$\mapsto$	$\texttt{"like"} \cdot \mathcal{L}_{syn}[d]$
$\mathcal{L}_{syn}[cl \centerdot d_1 \centerdot d_2]$	$\mapsto$	$\mathcal{L}_{syn}[d_1] \cdot \mathcal{L}_{syn}[d_2]$

Table 1. $\mathcal{L}_{syn}$ as a term re-writing system

T_A terms to the terms of linear simply-typed lambda-calculus with T_S terms as constants: Table 2. Applying such $\mathcal{L}_{syn}$ to the original

$\mathcal{L}_{syn}[boy]$	$\mapsto$	`"boy"`
$\mathcal{L}_{syn}[girl]$	$\mapsto$	`"girl"`
$\mathcal{L}_{syn}[a]$	$\mapsto$	$\lambda d.\, \texttt{"a"} \cdot d$
$\mathcal{L}_{syn}[every]$	$\mapsto$	$\lambda d.\, \texttt{"every"} \cdot d$
$\mathcal{L}_{syn}[like]$	$\mapsto$	$\lambda d.\, \texttt{"like"} \cdot d$
$\mathcal{L}_{syn}[cl]$	$\mapsto$	$\lambda d_1 d_2.\, d_1 \cdot d_2$

Table 2. $\mathcal{L}_{syn}$ as a mapping to lambda-calculus

ex_{bg} gives a lambda-expression, which, upon normalization, becomes the desired T_s term. The lambda-calculus view of $\mathcal{L}_{syn}$ goes back to the original ACG [1]. It is a good way of mechanically implementing the transformation, and used in our semantic calculator. After all, lambda-calculus is a term-rewriting system. Nevertheless, this view is “too concrete”, showing implementation details like the normalization step. The term re-writing–system view, shown in Table 1, offers the right amount of abstraction. Therefore, it will be often used throughout the paper.

There is yet another way to look at $\mathcal{L}_{syn}$: Table 3 is derived from Table 1 by replacing each $\mathcal{L}_{syn}[d]$ expression with the type of the T_A term d. The result looks like a context-free grammar. Therefore, the abstract term ex_{bg} can be viewed as a parsed tree of the grammar,

N	$\mapsto$	`"boy"`
N	$\mapsto$	`"girl"`
NP	$\mapsto$	`"a"` $\cdot N$
NP	$\mapsto$	`"every"` $\cdot N$
VP	$\mapsto$	`"like"` $\cdot NP$
S	$\mapsto$	$NP \cdot VP$

Table 3. $\mathcal{L}_{syn}$ as a context-free grammar

and $\mathcal{L}_{syn}$ as computing its yield. There is clearly an inverse transformation $\mathcal{L}_{syn}^{-1}$ – parsing from the surface form T_S to the parse tree T_A. The grammar in Table 3 is rather simple, thanks to the simple, unlifted types of the quantifying determiners like *every*.

The meaning of ex_{bg} is derived by applying a sequence of other transformations to it. The first transformation $\mathcal{L}_U$ turns ex_{bg} to a term in a language $T_A \cup \{a, var, U\}$, with new constants in place of *every*.

$$\begin{array}{rcl} \mathcal{L}_U[cl \centerdot C[every \centerdot d_r] \centerdot d] & \mapsto & (U_x \centerdot d_r) \centerdot (cl \centerdot C[var_x] \centerdot d) \\ \mathcal{L}_U[cl \centerdot d \centerdot C[every \centerdot d_r]] & \mapsto & (U_x \centerdot d_r) \centerdot (cl \centerdot d \centerdot C[var_x]) \end{array}$$

where $C[]$ is a context (a term with a hole) such that the hole is not a subterm of a cl term. §4 will show a few more rules for $\mathcal{L}_U$. If none of them apply, $\mathcal{L}_U$ is the identity. The result of $\mathcal{L}_U[ex_{bg}]$

$$(U_x \centerdot boy) \centerdot (cl \centerdot var_x \centerdot (like \centerdot (a \centerdot girl)))$$

is in effect the Quantifier Raising (QR) of "every boy", but in a rigorous, deterministic way. The intent of the new constants should become clear: U is to represent the raised quantifier, and var its trace. Unlike QR, the raised quantifier $(U_x \centerdot boy)$ lands not just on any suitable place. $\mathcal{L}_U$ puts it at the closest boundary marked by the clause-forming constant cl. It should also be mentioned that $\mathcal{L}_U$, as $\mathcal{L}_{sym}$, is type-preserving: it maps a well-typed term to also a well-typed term. The type preservation is the necessary condition for the correctness of the transformations. Again unlike QR, we specify the correctness conditions precisely.

The analogous $\mathcal{L}_E$ transformation raises "a girl", turning the above $T_A \cup \{a, var, U\}$ term into a term in the language $T_A \cup \{var, U, E\}$, without the constant a and with new constants E:

$$(U_x \centerdot boy) \centerdot ((E_y \centerdot girl) \centerdot (cl \centerdot var_x \centerdot (like \centerdot var_y)))$$

There are no longer any of the original quantifiers. The raised existential is placed at the boundary marked by cl.

$\mathcal{L}_{sem}[boy]$	$\mapsto$	boy
$\mathcal{L}_{sem}[girl]$	$\mapsto$	girl
$\mathcal{L}_{sem}[var_x]$	$\mapsto$	x
$\mathcal{L}_{sem}[(U_x \bullet d_r) \bullet d]$	$\mapsto$	$\forall x.\, \mathcal{L}_{sem}[d_r] \Rightarrow \mathcal{L}_{sem}[d]$
$\mathcal{L}_{sem}[(E_x \bullet d_r) \bullet d]$	$\mapsto$	$\exists x.\, \mathcal{L}_{sem}[d_r] \wedge \mathcal{L}_{sem}[d]$
$\mathcal{L}_{sem}[like \bullet d]$	$\mapsto$	$\mathsf{like}\ \mathcal{L}_{sem}[d]$
$\mathcal{L}_{sem}[cl \bullet d_1 \bullet d_2]$	$\mapsto$	$\mathcal{L}_{sem}[d_2]\ \mathcal{L}_{sem}[d_1]$

Table 4. $\mathcal{L}_{sem}$ as a term re-writing system

The final, straightforward transformation $\mathcal{L}_{sem}$, Table 4, turns the above term into the T_L logical formula $\forall x.\, \mathsf{boy}\ x \Rightarrow \exists y.\, \mathsf{girl}\ y \wedge \mathsf{like}\ y\ x$ representing the meaning of the original sentence. If $\mathcal{L}_E$ is applied first and $\mathcal{L}_U$ second, the resulting logical formula will have the opposite order of quantifiers, denoting the inverse reading of the sentence. The quantifier ambiguity hence comes from the order of applying individual transformations.

When a sentence has several quantifying determiners, each of them can be associated with its own transformation $\mathcal{L}$ that will eliminate, or raise them. Although there can be many ways to order such transformations, not every order will give a distinct reading, as we shall see soon.

The quantifier ambiguity was also treated in ACG [3], but differently, using quantifier lowering, which required guessing the (parsed) abstract form with the raised quantifiers. On our analysis, the parsed form has quantifiers in-situ, represented by the simple second-order ACG. No guessing of abstract forms is required.

2.2 Scope Islands

In the case of a scope island, like in "That every boy left upset a teacher", the abstract form

$$cl \bullet (that \bullet (cl \bullet (every \bullet boy) \bullet left)) \bullet (upset \bullet (a \bullet teacher))$$

has two cl constants corresponding to the subordinate and matrix clauses. The former is the closest to "every boy" and becomes the landing place for the raised universal, which is hence confined to the subordinate clause. As the result, changing the order of $\mathcal{L}_U$ and $\mathcal{L}_E$ transformations does not change the resulting logic formula: the original sentence does not have the quantifier ambiguity.

2.3 Lexicon: Term-Language Transformation

The transformations like $\mathcal{L}_U$, $\mathcal{L}_{syn}$, etc. (called 'lexicon') have been specified as type-preserving and confluent term-rewriting systems. As we have already mentioned for $\mathcal{L}_{syn}$, lexicon can also be programmed in linear lambda-calculus. After all, lambda-calculus is a type-preserving and confluent term-rewriting system. To transform a T_1 term into a term in the language T_2, we replace each constant in the original T_1 term by a λ-expression $\Lambda(T_2)$ and normalize the result. Here, $\Lambda(T_2)$ means a typed lambda-calculus with sums, products and the fixpoint whose constants are terms of T_2. If the normalization succeeds, we end up with the term T_2, the result of the transformation. Because of the fixpoint, there may be no normal form: the term transformations are inherently partial.

The transformations $\mathcal{L}_{syn}$ and $\mathcal{L}_{sem}$ are clearly simple homorphisms. $\mathcal{L}_U$ and $\mathcal{L}_E$ are intuitively understood as movements, to a precisely-defined boundary. The exact details of the lambda-calculus implementation, fully described in [4], are not needed to understand the present paper or to use AACG. Often the term-rewriting system view is the right level of abstraction.

3 Anaphora and the Modeling of Dynamic Semantics

This section describes AACG analyses of binding and its interaction with other phenomena, in particular, quantification and scoping islands. We start with the trivial example "Mary loves her mother", whose abstract form is as follows:

$$cl \bullet Mary \bullet (love \bullet (possess \bullet she \bullet mother))$$

It is written in the language $T_A \cup \{pronoun\}$; the constant *possess* has the type $NP \rightarrowtail N \rightarrowtail NP$.

We now describe in detail the transformation $\mathcal{L}_{dyn}$ that will eliminate the pronoun, replacing it with its referent. The transformation is performed in two phases. The preliminary phase adds annotations to the abstract form, to be eliminated, along with the pronouns, by the $\mathcal{L}_{dyn}$ proper. The annotation phase $\mathcal{L}_{ann}$ is a lexicon transformation like before, into a T_A language enriched with two additional constants

$$\begin{aligned} \mathit{update}\colon\ & \mathit{Gender} \rightarrowtail NP \rightarrowtail \sigma \rightarrowtail \sigma \\ \mathit{post}\colon\ & NP \rightarrowtail NP \end{aligned}$$

The annotation transformation is

$$\begin{array}{lcl} \mathcal{L}_{ann}[\mathit{Mary}] & \mapsto & \mathit{post} \bullet (\mathit{update} \bullet \mathit{Fem} \bullet \mathit{Mary} \bullet \mathit{Mary}) \\ \mathcal{L}_{ann}[\mathit{var}_x] & \mapsto & \mathit{post} \bullet \mathit{var}_x \\ \mathcal{L}_{ann}[(U_x \bullet \mathit{girl}) \bullet d] & \mapsto & (U_x \bullet (\mathit{update} \bullet \mathit{Fem} \bullet \mathit{var}_x \bullet \mathit{girl})) \bullet \mathcal{L}_{ann}[d] \end{array}$$

It is the identity otherwise. Applying $\mathcal{L}_{ann}$ to or example gives

$$\begin{gathered} \mathit{cl} \bullet (\mathit{post} \bullet (\mathit{update} \bullet \mathit{Fem} \bullet \mathit{Mary} \bullet \mathit{Mary})) \bullet \\ (\mathit{love} \bullet (\mathit{possess} \bullet \mathit{she} \bullet \mathit{mother})) \end{gathered}$$

The transformation $\mathcal{L}_{dyn}$ will try to eliminate the pronoun such as *she*, replacing it with with a post-ed term. (And the follow-up trivial transformation will erase the no-longer needed *post* and *update* annotations). It is easier to explain the context-sensitive re-writing $\mathcal{L}_{dyn}$ using the notion of a "discourse context" – which is the global state maintained by $\mathcal{L}_{dyn}$ as it traverses the term in-order: left-to-right, depth-first. When $\mathcal{L}_{dyn}$ encounters $\mathit{post} \bullet d$, it posts d into the discourse context. The annotation $\mathit{update} \bullet \mathit{Fem} \bullet d_1 \bullet d_2$ represents d_2 while recording a constraint in the context: d_1 has feminine gender. The *update* annotation does not post the term; it merely records the constraint. $\mathcal{L}_{dyn}[\mathit{she}]$ searches the context for a posted term associated with the feminine gender, and returns that term. In our example, $(\mathit{post} \bullet (\mathit{update} \bullet \mathit{Fem} \bullet \mathit{Mary} \bullet \mathit{Mary}))$ records the constraint that *Mary* has feminine gender, and then posts *Mary* into the context, where *she* will find it. In the result, $\mathcal{L}_{dyn}$ gives

$$\mathit{cl} \bullet \mathit{Mary} \bullet (\mathit{love} \bullet (\mathit{possess} \bullet \mathit{Mary} \bullet \mathit{mother}))$$

which no longer has any pronouns.

If we replace "she" with "he" in the original sentence, $\mathcal{L}_{dyn}[\mathit{he}]$ will fail to find the suitable referent (assuming the initially empty discourse context). The failure means the $\mathcal{L}_{dyn}$ transformation was unsuccessful and hence the sentence corresponding to the source term is unacceptable. This negative prediction is not delivered in [6] or ACG analyses since they not make gender distinctions.

$\mathcal{L}_{dyn}$ may be implemented using a global mutable state, state monad, delimited continuations, and so on – or as a context-sensitive term re-writing system. Either way, the implementation details should not matter. What matters is that $\mathcal{L}_{dyn}[she]$ examines the parents and the left siblings of *she*, looking for a posted term of the feminine gender.

The transformation $\mathcal{L}_{dyn}$ easily composes with $\mathcal{L}_U$ and $\mathcal{L}_E$, giving an account of quantification and binding, as in

(1) Every girl$_i$'s father loves her$_i$ mother.
(1b) *Her$_i$ father loves every girl$_i$'s mother.
(2a) That John$_i$ left upset his$_i$ teacher.
(2b) *That every boy$_i$ left upset his$_i$ teacher.

To wit, the abstract form of (1), written in the language $T_A \cup \{pronoun, every\}$ is

$$cl \centerdot (possess \centerdot (every \centerdot girl) \centerdot father) \centerdot (love \centerdot (possess \centerdot she \centerdot mother))$$

The transformation $\mathcal{L}_{sem}$ that produces the logic formula does not apply since that lexicon has no mapping for pronouns and *every*. They have to be eliminated first, by applying $\mathcal{L}_U$ followed by $\mathcal{L}_{dyn}$. First, $\mathcal{L}_U$ with $\mathcal{L}_{ann}$ produce

$$\begin{aligned}&(U_x \centerdot (update \centerdot Fem \centerdot var_x \centerdot girl)) \centerdot\\ &\quad (cl \centerdot (possess \centerdot (post \centerdot var_x) \centerdot father) \centerdot\\ &\qquad (love \centerdot (possess \centerdot she \centerdot mother)))\end{aligned}$$

Therefore, $\mathcal{L}_{dyn}$ will first record the constraint that var_x is feminine, then post var_x, and then find it when encountering *she*. The result is

$$(U_x \centerdot girl) \centerdot (cl \centerdot (possess \centerdot var_x \centerdot father) \centerdot (love \centerdot (possess \centerdot var_x \centerdot mother)))$$

with the eliminated (resolved) pronoun. $\mathcal{L}_{sem}$ can now derive the logical formula representing the meaning of the sentence.

From the left-to-right, depth-first traversal mode of $\mathcal{L}_{dyn}$ we predict that a referent for a pronoun in an acceptable sentence must be located to the 'left' of the pronoun, i.e., earlier in the in-order tree traversal. For example, we predict that (1b) is unacceptable. The scoping island condition for the universal then explains the unacceptability of (2b).

Our prediction may seem at odds with (3a) and (3b):

(3a) Alice's present for him$_i$, every boy$_i$ saw.
(3b) *Every boy$_i$, his$_i$ mother likes.

Looking at the abstract form of (3b), for example,

$$(\mathit{frontNP} \centerdot (\mathit{every} \centerdot \mathit{boy})) \centerdot (\mathit{cl} \centerdot (\mathit{possess} \centerdot \mathit{he} \centerdot \mathit{mother}) \centerdot (\mathit{like} \centerdot \_\_))$$

we see the constant *frontNP* that corresponds to the comma that sets off the fronted NP. $\mathcal{L}_{sem}$ has no mapping for that constant and for the trace $\_\_$, therefore, they have to be transformed away first. Our transformation stack will hence begin with $\mathcal{L}_{lower}$ that lowers the fronted phrase into the position indicated by $\_\_$. The other transformations run afterwards. By the time of $\mathcal{L}_{dyn}$, the variable representing the moved out $\mathit{every} \centerdot \mathit{boy}$ will be to the right of *he* and so the pronoun cannot refer to it.

We finish with the interesting example

(1c) A girl$_i$ met every boy who liked$_i$ her.

Although it has an existential and a universal QNPs, it exhibits no quantifier ambiguity. To understand why, consider its abstract form

$$\mathit{cl} \centerdot (a \centerdot \mathit{girl}) \centerdot (\mathit{met} \centerdot (\mathit{every} \centerdot (\mathit{who} \centerdot \mathit{boy} \centerdot (\mathit{liked} \centerdot \mathit{she}))))$$

where the constant *who* has the type $N \rightarrowtail VP \rightarrowtail N$. Applying $\mathcal{L}_E$ with $\mathcal{L}_{ann}$ (and omitting *update* for clarity) gives

$$(E_x \centerdot \mathit{girl}) \centerdot (\mathit{cl} \centerdot (\mathit{post} \centerdot \mathit{var}_x) \centerdot (\mathit{met} \centerdot (\mathit{every} \centerdot (\mathit{who} \centerdot \mathit{boy} \centerdot (\mathit{liked} \centerdot \mathit{she})))))$$

If the universal is raised first, the result

$$(U_y \centerdot (\mathit{who} \centerdot \mathit{boy} \centerdot (\mathit{liked} \centerdot \mathit{she})))((E_x \centerdot \mathit{girl}) \centerdot (\mathit{cl} \centerdot (\mathit{post} \centerdot \mathit{var}_x) \centerdot (\mathit{met} \centerdot \mathit{var}_y)))$$

clearly has the pronoun *she* to the right of its posted referent and hence cannot be bound by it.

4 Inverse Linking

This section describes the analyses of inverse linking (4a-4c):

(4a) At least two senators on every committee voted against the bill.
(4b) Two politicians spy on someone from every city.
(4c) Some man from every city secretly despises it.

The examples, borrowed from [5] demonstrate three characteristic features of this phenomenon. First, a QNP embedded into a prepositional phrase attached to a noun of another QNP takes scope over that outer noun phrase: (4a) has a reading with "every committee" taking scope over "the two senators". The second feature of inverse linking is that an external QNP like "two politicians" in (4b) cannot scope between the inversely linked "every" and "someone". Lastly, the embedded QNP on the inverse linking reading may bind pronouns in other parts of the sentence.

Previously, the restrictor of a QNP was rather simple, often merely a common noun. Inverse linking is about QNPs whose restrictor itself contains a QNP, which requires generalization of our $\mathcal{L}_U$ and $\mathcal{L}_E$ transformations. To see why, consider what happens if we apply the existing $\mathcal{L}_E$ to (4b):

$$\begin{array}{l}(E_x \mathbin{\centerdot} (\mathit{from} \mathbin{\centerdot} \mathit{person} \mathbin{\centerdot} (\mathit{every} \mathbin{\centerdot} \mathit{city}))) \mathbin{\centerdot} \\ (\mathit{cl} \mathbin{\centerdot} (\mathit{two} \mathbin{\centerdot} \mathit{politician}) \mathbin{\centerdot} (\mathit{spyOn} \mathbin{\centerdot} \mathit{var}_x))\end{array}$$

We have to eliminate *every* since $\mathcal{L}_{sym}$ has no mapping for it. However, $\mathcal{L}_U$ cannot apply to $(\mathit{every} \mathbin{\centerdot} \mathit{city})$ since that subterm does not appear within the *cl* context.

We have to generalize the quantifier movement transformations, and take restrictors seriously. First, we re-define U_x and E_x constants and introduce the explicit constant for restrictors:

$$\begin{array}{rl} U_x, U_y, \ldots : & S \rightarrowtail S \rightarrowtail S \\ E_x, E_y, \ldots : & S \rightarrowtail S \rightarrowtail S \\ \mathit{restr} : & N \rightarrowtail NP \rightarrowtail S \end{array}$$

The new $\mathcal{L}_U$ is as follows:

$$\begin{array}{ll}
(1) & \mathcal{L}_U[\mathit{cl} \mathbin{\centerdot} C[\mathit{every} \mathbin{\centerdot} d_r] \mathbin{\centerdot} d] \quad \mapsto \\
 & \qquad (U_x \mathbin{\centerdot} (\mathit{restr} \mathbin{\centerdot} d_r \mathbin{\centerdot} \mathit{var}_x)) \mathbin{\centerdot} (\mathit{cl} \mathbin{\centerdot} C[\mathit{var}_x] \mathbin{\centerdot} d) \\
(2) & \mathcal{L}_U[\mathit{cl} \mathbin{\centerdot} d \mathbin{\centerdot} C[\mathit{every} \mathbin{\centerdot} d_r]] \quad \mapsto \\
 & \qquad (U_x \mathbin{\centerdot} (\mathit{restr} \mathbin{\centerdot} d_r \mathbin{\centerdot} \mathit{var}_x)) \mathbin{\centerdot} (\mathit{cl} \mathbin{\centerdot} d \mathbin{\centerdot} C[\mathit{var}_x]) \\
(3) & \mathcal{L}_U[\mathit{restr} \mathbin{\centerdot} C[\mathit{every} \mathbin{\centerdot} d_r] \mathbin{\centerdot} \mathit{var}_x] \quad \mapsto \\
 & \qquad (U_y \mathbin{\centerdot} (\mathit{restr} \mathbin{\centerdot} d_r \mathbin{\centerdot} \mathit{var}_y)) \mathbin{\centerdot} (\mathit{restr} \mathbin{\centerdot} C[\mathit{var}_y] \mathbin{\centerdot} \mathit{var}_x) \\
(4) & \mathcal{L}_U[U_x \mathbin{\centerdot} (\mathit{restr} \mathbin{\centerdot} C[\mathit{every} \mathbin{\centerdot} d_r] \mathbin{\centerdot} \mathit{var}_x) \mathbin{\centerdot} d] \quad \mapsto \\
 & \qquad (U_y \mathbin{\centerdot} (\mathit{restr} \mathbin{\centerdot} d_r \mathbin{\centerdot} \mathit{var}_y)) \mathbin{\centerdot} (U_x \mathbin{\centerdot} (\mathit{restr} \mathbin{\centerdot} C[\mathit{var}_y] \mathbin{\centerdot} \mathit{var}_x) \mathbin{\centerdot} d) \\
(5) & \mathcal{L}_U[E_x \mathbin{\centerdot} (\mathit{restr} \mathbin{\centerdot} C[\mathit{every} \mathbin{\centerdot} d_r] \mathbin{\centerdot} \mathit{var}_x) \mathbin{\centerdot} d] \quad \mapsto \\
 & \qquad (U_y \mathbin{\centerdot} (\mathit{restr} \mathbin{\centerdot} d_r \mathbin{\centerdot} \mathit{var}_y)) \mathbin{\centerdot} (E_x \mathbin{\centerdot} (\mathit{restr} \mathbin{\centerdot} C[\mathit{var}_y] \mathbin{\centerdot} \mathit{var}_x) \mathbin{\centerdot} d)
\end{array}$$

The hole in the context $C[]$ should not appear as a sub-term of *cl*, *restr*, or within a QNP.

We now show the analysis of (4b) with the generalized quantifier movements. First we observe that we cannot move out $\mathit{every} \mathbin{\centerdot} \mathit{city}$ from the original term since this subterm is part of larger QNP. Recall that the hole in the context $C[]$ must not appear within a QNP. Suppose we first apply $\mathcal{L}_E$, producing

$$(E_x \centerdot (restr \centerdot (from \centerdot person \centerdot (every \centerdot city)) \centerdot var_x)) \centerdot$$
$$(cl \centerdot (two \centerdot politician) \centerdot (spyOn \centerdot var_x))$$

There are now two choices of applying the $\mathcal{L}_U$ transformation. The first choice, using rule (3) of the new $\mathcal{L}_U$ transformation, gives

$$(E_x \centerdot$$
$$((U_y \centerdot (restr \centerdot city \centerdot var_y)) \centerdot (restr \centerdot (from \centerdot person \centerdot var_y) \centerdot var_x))) \centerdot$$
$$(cl \centerdot (two \centerdot politician) \centerdot (spyOn \centerdot var_x))$$

with "someone" scoping over "every city". The other choice, using rule (5) of $\mathcal{L}_U$, produces

$$((U_y \centerdot (restr \centerdot city \centerdot var_y)) \centerdot$$
$$(E_x \centerdot (restr \centerdot (from \centerdot person \centerdot var_y) \centerdot var_x))) \centerdot$$
$$(cl \centerdot (two \centerdot politician) \centerdot (spyOn \centerdot var_x))$$

with the wide-scoping "every city". This is the case of inverse linking. In either case, "two politicians" takes the narrowest scope. If "two politicians" is moved out first, it takes the sentence-wide scope. In no case this QNP can scope between "someone" and "every city", reproducing the empirical restriction.

5 Conclusions

We have demonstrated AACG as the framework for uniform analyses of variety of phenomena related to quantification and binding, including the rarely treated (outside dynamic semantics) the gender marking of pronouns. The key idea is successive transformations of an abstraction of the syntactic form until we obtain the logic formula representing its meaning. We specifically used the examples of [6] to contrast the AACG's take on linguistic side-effects, effect interactions and the notion of evaluation. Whereas in [6] all linguistic side-effects (movements) occur during the single tree traversal (evaluation), we decompose them into separate simple traversals. AACG takes compositionality to a new level, not only of meanings but transformations.

Linguistic side-effects use the single delimited control operator shift for everything: to implement the discourse context and the movements similar to $\mathcal{L}_U$. The versatility had the price of rigidity. Quantification ambiguity could only be realized by changing the global evaluation order, which affects all other predictions. Delimited control is also a low-level implementation detail. In our approach, the transformations are specified rather abstractly, as type-preserving term rewriting. Although each individual lexicon transformation is confluent, there is a choice in their ordering, giving enough flexibility for individual movements – with rigidity to reproduce the scoping restrictions of inverse linking.

The present paper talked entirely about semantic interpretation of sentences, or parsing a sentence to a logical form, so to speak. We have said nothing about the converse, finding a sentence whose meaning matches the given logical form. That is, we have been solving the problem of comprehension and have not at all investigated generation. Although using arbitrary applicative functors clearly makes the generation problem intractable, there could be constraints imposed on the applicatives that make the 'inversion' of their transformations tractable, like the almost linear constraint of [2]. Viewing generation as logic programming problem, as did [2], seems promising.

References

[1] de Groote, P.: Towards abstract categorial grammars. In: Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics. pp. 148–155. Morgan Kaufmann, San Francisco, CA (Jul 2002), `http://www.aclweb.org/anthology/P01-1033`

[2] Kanazawa, M.: Parsing and generation as Datalog query evaluation (2011), `\url{http://research.nii.ac.jp/~kanazawa/publications/pagadqe.pdf}`

[3] Kanazawa, M., Pogodalla, S.: Advances in Abstract Categorial Grammars: Language theory and linguistic modeling. Lecture notes, ESSLLI 09. Part 2 (Jul 2009), `http://www.loria.fr/equipes/calligramme/acg/publications/esslli-09/2009-esslli-acg-week-2-part-2.pdf`

[4] Kiselyov, O.: Applicative abstract categorial grammar. In: Kanazawa, M., Moss, L.S., de Paiva, V. (eds.) NLCS'15. Third Workshop on Natural Language and Computer Science, EasyChair Proceedings in Computing, vol. 32, pp. 29–38. EasyChair (2015)

[5] Kobele, G.M.: Inverse linking via function composition. Natural Language Semantics 18(2), 183–196 (2010)

[6] Shan, C.c.: Linguistic side effects. In: Barker, C., Jacobson, P. (eds.) Direct Compositionality. pp. 132–163. Oxford University Press, New York (2007)