Toward a Comprehensive Model of Grammar Based on the Typed Lambda Calculus

Makoto Kanazawa and Ryo Yoshinaka

Introduction

The Abstract Categorial Grammar (de Groote 2001) is a grammar formalism based on linear typed λ-calculus. It generalizes both string grammars and tree grammars by using linear λ-terms to represent strings, trees, grammatical derivations, etc. Our aim is to use the ACG to build a formal model of human grammar that is comprehensive in the sense that it is equipped with theories and practical algorithms for parsing, generation, and learning.

Strings as linear λ-terms

Fix an atomic type a, and let

\[ s_{tr} : a \rightarrow a. \]

To represent strings over an alphabet \( V \), introduce a constant c of type \( s_{tr} \) for each symbol \( v \) in \( V \). Let

\[ /c_1 \ldots c_n / = \lambda x^{n+1}. c_1(\ldots(c_n x)\ldots). \]

Writing + as an infix operator, we then have

\[ /w/+/w/ \rightarrow_{\beta} /wu/. \]

ACGs

An ACG \( \mathcal{G} \) consists of a set \( C \) of typed abstract constants and a mapping \( \mathcal{L} \) that associates each abstract constant in \( C \) with a closed linear λ-term (object term). There must be a type substitution \( \sigma \) that sends the type of \( c \) to the type of \( \mathcal{L}(c) \).

The abstract language \( \mathcal{A}(\mathcal{G}) \) of \( \mathcal{G} \) is the set of closed linear λ-terms built up from \( C \) that are of distinguished type \( s \). The object language \( \mathcal{O}(\mathcal{G}) \) of \( \mathcal{G} \) is the image of \( \mathcal{A}(\mathcal{G}) \) under \( \mathcal{L} \).

Complexity and generative capacity

The universal membership problem \( M \in \mathcal{O}(\mathcal{G}) \) is at least EXPSPACE-hard, but it is not known whether it is decidable. We have shown that the universal membership problem for lexicalized ACGs is NP-complete (Yoshinaka and Kanazawa 2005).

de Groote and Pogodalla (2004) showed that the class of languages generated by linear context-free rewriting systems (a subclass of PTIME consisting only of semilinear languages) is included in the class of string languages of ACGs. We have shown that the string languages of lexicalized ACGs form a subclass of NP which contains some NP-complete as well as non-semilinear languages and includes all the \( \epsilon \)-free languages generated by linear context-free rewriting systems (Yoshinaka and Kanazawa 2005).

We have also shown that the string languages of ACGs form a substitution-closed full AFL, while the string languages of lexicalized ACGs form a substitution-closed AFL (Kanazawa 2005a).

2-dimensional ACGs

A 2-dimensional ACG is a pair of ACGs \( \mathcal{G}_1, \mathcal{G}_2 \) sharing the same abstract vocabulary: A 2-dimensional ACG \( \mathcal{G} \) defines a relation

\[ \mathcal{R}(\mathcal{G}) = \{ (\mathcal{L}(G_1), \mathcal{L}(G_2)) \mid G \in \mathcal{A}(\mathcal{G}) \}. \]

If members of \( \mathcal{O}(\mathcal{G}_1) \) represent strings and members of \( \mathcal{O}(\mathcal{G}_2) \) represent meanings, \( \mathcal{R}(\mathcal{G}) \) represents the relation “string \( w \) means \( m \).”

Example: A small fragment of English

<table>
<thead>
<tr>
<th>abstract constant</th>
<th>type</th>
<th>object term</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>( [p_1 \rightarrow s] \rightarrow s )</td>
<td>( \lambda x^{n+1}. a[k_1^{n+1}/k_1] )</td>
</tr>
<tr>
<td>( b )</td>
<td>( [p_1 \rightarrow s] \rightarrow s )</td>
<td>( \lambda x^{n+1}. b[k_1^{n+1}/k_1] )</td>
</tr>
<tr>
<td>( c )</td>
<td>( q \rightarrow s )</td>
<td>( \lambda x^{n+1}. c[k_1^{n+1}/k_1] )</td>
</tr>
<tr>
<td>( d )</td>
<td>( q \rightarrow s )</td>
<td>( \lambda x^{n+1}. d[k_1^{n+1}/k_1] )</td>
</tr>
<tr>
<td>( e )</td>
<td>( p_1 \rightarrow p_2 \rightarrow p_3 \rightarrow q )</td>
<td>( \lambda x^{n+1}. e[k_1^{n+1}/k_1] )</td>
</tr>
<tr>
<td>( f )</td>
<td>( q \rightarrow q )</td>
<td>( \lambda x^{n+1}. f[k_1^{n+1}/k_1] )</td>
</tr>
</tbody>
</table>

\( \mathcal{O}(\mathcal{G}) = \{ \omega \mid \omega \in \text{MIX} \} \), where

\( \text{MIX} = \{ w \mid w \in [a, b, c, d, e, f]. \} \).

Parsing and generation

The problems of parsing (determining the meaning(s) of a given string) and of generation (putting a meaning into a string) for 2-dimensional ACGs are closely related to linear higher-order matching, which is an NP-complete problem and remains NP-complete even in a very restricted form (Yoshinaka 2005). Our collaborator Sylvain Salvati (2005) studied algorithms for linear higher-order matching and ACG parsing.

Learning

We take learning to be the problem of finding an adequate grammar from a (sufficiently large) finite set of string-meaning pairs. We assume that the target grammar is a lexicalized ACG in which the form associated with each abstract constant contains exactly one constant. An abstract constant, together with its form and meaning, then represents a lexical entry. Our approach to learning is to first identify the meaning of each lexical entry, and then use that to identify its form.

The first task, the learning of semantics, in turn consists of two stages, following Siskind’s (1996) method for learning lexical semantics expressed by first-order terms. In the first stage, we identify the set of constants involved in the meaning of each lexical entry. The second stage, where we build a meaning term using the set of constants identified in the first stage, requires a significant extension of Siskind’s method to our higher-order setting. We use an algorithm for computing interpolants in implicational logics (Kanazawa 2005b) to solve this problem.

When we have a string-meaning pair in which the meanings of all relevant lexical entries have been correctly identified, we can determine the abstract derivation for the pair using the principal-type algorithm. Identifying the correct form of each lexical entry then becomes a matter of solving a certain matching equation.