Combinatory Logic and Natural Language Parsing

Cem Bozşahin
Department of Computer Engineering
Middle East Technical University
06531, Ankara
bozsahin@ceng.metu.edu.tr

Abstract

We describe the connections between the primitives of Combinatory Logic and operations in natural language syntax. We also show how word order variation in Turkish syntax can be explained by a few primitives of Combinatory Logic. A computational framework for Turkish syntax for parsing surface structures into combinator expressions is outlined. Evaluation of combinator expressions (semantic forms) has been shown to be similar to interpreting functional programming languages.

This research is supported in part by grants from TÜBİTAK (project no. EEEAG-90) and NATO TU-LANGUAGE Project.

tel: (312)210-5580
fax: (312)210-1259

1 Introduction

Formal approaches to natural language syntax and semantics have their roots in revolutionary developments in the early decades of the Twentieth Century. Frege and Husserl's view of language as a system of categories, Russell's theory of types, Church's λ-calculus, Carnap's formal logic, Tarski's work on model theory, and Curry's combinatory logic are some of the milestones of these developments. These advances in formal sciences have had important influence on the birth of formal approaches to linguistics, as witnessed by Bar-Hillel's [1], Chomsky's [2, 3, 4], Lambek's [5], and Montague's [6] works.

The lineage from Frege, Curry, Lambek, Bar-Hillel to Montague is of particular interest here due to an emerging algebraic view of language as a system of relations, and linguistics as a formal study of these relations. Along with the works of Lesniewski and Ajdukiewicz in the 1930s, these developments led to what is now known as Categorial Grammar (CG), a linguistic theory of grammar with firm foundations in logic. Ajdukiewicz and Bar-Hillel may be considered the forefathers of using CG as a linguistic methodology in the analysis of language. In their view, linguistic expressions can be interpreted as functors, arguments, and application of functors to their arguments. For instance, determiners in English can be seen as functors that take a noun (N) and produce a noun phrase (NP) as a result:

$$Det : N \mapsto NP \qquad \text{or,} \qquad Det(N) = NP$$

Ajdukiewicz used the order-neutral notation $\frac{NP}{N}$ or NP/N where the numerator is the result and the denominator is the argument. But natural language syntax is sensitive to the relative order of functors and arguments, hence an order-dependent interpretation of NP/N is needed; it is treated as a functor which yields NP if an N is found to the right of the functor: [1]

$$\frac{\begin{array}{cc} \textit{the} & \textit{man} \\ NP/N & N \end{array}}{NP} \qquad\qquad \frac{\begin{array}{cc} *\textit{man} & \textit{the} \\ N & NP/N \end{array}}{*}$$

Similarly, there is the backward-looking variety of functors, as the following example from Turkish illustrates:

[1] We use the common practice in linguistics of marking ill-formed or ungrammatical examples with a '*'

$$\begin{array}{c} \textit{adam} \quad \textit{uyu-du} \\ \text{man.NOM sleep-TENSE} \\ \textit{NP} \quad \textit{S}\backslash\textit{NP} \\ \hline \textit{S} \end{array}$$

'The man slept'

S is the category of predicates (propositions). Infinitely many categories can be obtained by the following recursive definition of well-formed categories:

- *S, NP* and *N* are (basic) categories.
- if X and Y are categories, then X/Y, $X\backslash Y$ and *(X)* are also (derived) categories.
- Nothing else is a category.

For instance, the transitive verbs of English have the category $(S\backslash NP)/NP$, i.e., to become a proposition, the verb must find an NP to the right (the object) and then an NP to the left (the subject). Formal definition of function application is given in (1); example (2) shows the use of these rules in derivation of linguistic expressions. The symbols f and a in (1) represent respectively the semantic representation of functor and the argument. The semantic object produced by function application is shown by juxtaposition, e.g., fa in (1).

(1) Forward Application ($\mathbf{A}_>$): $\begin{array}{ccc} X/Y & Y & \Rightarrow \quad X \\ f & a & \quad fa \end{array}$

Backward Application ($\mathbf{A}_<$): $\begin{array}{ccc} Y & X\backslash Y & \Rightarrow \quad X \\ a & f & \quad fa \end{array}$

(2)
$$\dfrac{\dfrac{\dfrac{\textit{the}}{\textit{NP}/\textit{N}} \quad \dfrac{\textit{man}}{\textit{N}}}{\textit{NP}}\mathbf{A}_> \quad \dfrac{\dfrac{\dfrac{\textit{read}}{(\textit{S}\backslash\textit{NP})/\textit{NP}} \quad \dfrac{\dfrac{\textit{the}}{\textit{NP}/\textit{N}} \quad \dfrac{\textit{book}}{\textit{N}}}{\textit{NP}}\mathbf{A}_>}{\textit{S}\backslash\textit{NP}}\mathbf{A}_>}{\textit{S}}\mathbf{A}_<$$

Taking function application as a primitive operation and incorporating Schönfinkel's [7] system into their logic, Curry and Feys [8] defined several other primitives of

function manipulation, called *combinators*, in a variable-free notation. The counterparts of these formal devices have been observed to be in widespread use in natural languages. This led to the development of Combinatory Categorial Grammar (CCG) [9, 10, 11]. Syntactic operations that are observed in natural language syntax formally reduce to a handful of primitive operations such as application, composition, swapping, duplication, etc. Their semantics also have a close parallel with the semantics of combinators. Steedman [10] provides ample reasons as to why combinators are preferable for natural language grammars. From a cognitive perspective, variable-less semantic interpretation seems a more plausible alternative for understanding human language faculty. Basically, the combinators allow for abstraction (as well as application) in semantic form without having to use variables. As variables introduce complications such as scope and binding (i.e., the environment), abstraction without variables purportedly puts less of a burden on processing due to lesser bookkeeping in a combinatory system. An analogy can be made here to similar reasons of economy in artificial languages: Turner [12] describes a combinatory system for purely applicative LISP in which all occurences of variables are removed during compilation so as to obtain an object code that can run more efficiently, since it does not require environment creation and deletion, a time consuming task at run-time. Extensive research on cognitive aspects of syntax-semantics interaction needs to be done to support the hypothesis and the analogy.

In the domain of natural languages, the combinator expressions (as semantic forms) reveal the predicate-argument structure of an utterance, in tandem with syntactic processing. This form can be evaluated to obtain predicate, subject, object etc. of an utterance. In other words, the presence of combinators in a semantic form indicates how this form is derived syntactically, thus making the syntax-semantics connection explicit.

In the remainder of the paper, we describe some of these devices and their use in Turkish syntax. We also describe a computational framework for Turkish syntax in which Turkish utterances are converted into a semantic representation in the form of combinator expressions. Evaluation of these expressions are also discussed.

2 Combinators in Natural Languages

An important aspect of Turkish syntax is that word order variation is allowed if the verb's arguments are case-marked for their syntactic function. Turkish is known as a subject-object-verb (SOV) language, but all six variations are possible if arguments are overtly marked. The basic word order SOV can be handled by application rules alone, if the category assignments in (3) are assumed for verbs and NPs. The subscripts on NPs do not represent new basic CG categories, but feature (attribute) marking of NPs on case. Computationally, this can be easily verified by using a unification-based feature checking system, as outlined in [13]. NOM, ACC and DAT are abbreviated feature values for nominative, accusative, and dative case, respectively. $|$ denotes a slash underspecified for directionality; it is instantiated to $\backslash$ or $/$ during rule matching. Underspecification may not be necessary in configurational languages such as English. It is required in Turkish only for post-verbal scrambling of the verb's arguments.

(3)

Ayşe	*kitab-ı*	*oku-du*
A.NOM	book-ACC	read-TENSE
NP_{nom}	NP_{acc}	$S\|NP_{nom}\|NP_{acc}$
	$S\|NP_{nom}$ (A<)	
	S (A<)	

'Ayşe read the book.'

However, if the subject is scrambled next to the verb, it is no longer possible for the verb to find its outermost argument NP_{acc} in the left-adjacent position:

(4)

kitab-ı	*Ayşe*	*oku-du*
NP_{acc}	NP_{nom}	$S\|NP_{nom}\|NP_{acc}$
	$S\|NP_{acc}$ (?)	

This problem occurs mainly because only verbs are treated as active elements (functors), and arguments are simply waiting to be picked up by the functors. If an argument can be turned into an active element as well, it can look for functors which require an argument of this type. This is precisely what is achieved by an operation called *type raising* (TR) in CG. Formally, it is defined as [14]:

(5) Type Raising (**T**): $\underset{a}{NP} \Rightarrow \underset{\lambda f.fa}{T/(T\backslash NP)}$ or $T\backslash(T/NP)$

There is another kind of operation whose usage is almost always intertwined with that of TR, namely, function composition. There are four variants of composition to take care of relative order of the principal functor (f) and the secondary functor (g) [9]:

(6) Forward Composition ($\mathbf{B}_{>}$): $\underset{f}{X/Y} \quad \underset{g}{Y/Z} \quad \Rightarrow \quad \underset{\lambda x.f(gx)}{X/Z}$

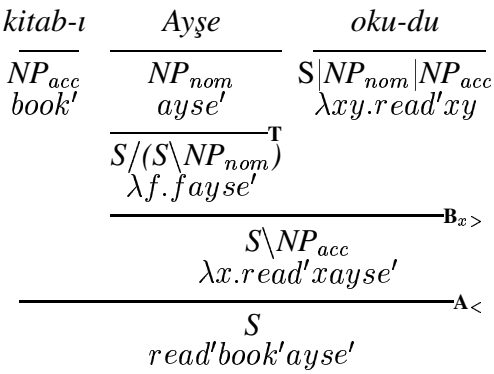
Backward Composition ($\mathbf{B}_{<}$): $\underset{g}{Y\backslash Z} \quad \underset{f}{X\backslash Y} \quad \Rightarrow \quad \underset{\lambda x.f(gx)}{X\backslash Z}$

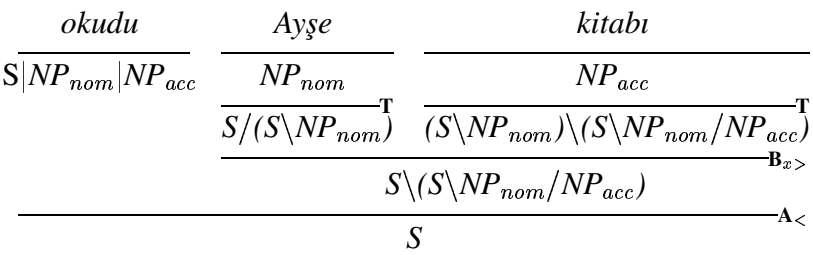
Forward Crossing Composition ($\mathbf{B}_{x>}$): $\underset{f}{X/Y} \quad \underset{g}{Y\backslash Z} \quad \Rightarrow \quad \underset{\lambda x.f(gx)}{X\backslash Z}$

Backward Crossing Composition ($\mathbf{B}_{x<}$): $\underset{g}{Y/Z} \quad \underset{f}{X\backslash Y} \quad \Rightarrow \quad \underset{\lambda x.f(gx)}{X/Z}$

With these operations, example (4) can be accounted for as in (7).[2] The order of lambda abstractions for *okudu* indicates the order in which the verb expects its arguments: first NP_{acc} then NP_{nom}, i.e., the slashes are left-associative, and $S|NP_{nom}|NP_{acc}$ is same as $(S|NP_{nom})|NP_{acc}$. In the lexicon, this sequence is specified for the canonical word order (SOV for Turkish), but since variations in word order do not change the predicate-argument structure of the verb, derivations using CG rules must select the correct argument for reduction, which is not always the outermost argument in the syntactic category. Other variants of SOV can be handled by specifying different values for T in (5). T seems to be quite language particular; we need T=$\{S, (S\backslash NP_{nom}), (S\backslash NP_{nom}\backslash NP_{dat})\}$ for Turkish to model all variants of word order and coordination. For instance, the second element is required for type-raising of the object *kitabı* in (8).

[2]Semantic forms of expressions are written underneath their syntactic category. In fact, CG categories are semantic categories in Husserl's sense. Primes indicate an abstraction of the semantic object whose internal structure is of no concern here. λ-calculus expressions are used for ease of exposition; they will be replaced by combinator expressions later on.

(7)


(8)


Type-raising and composition turns out to be equivalent to two of the combinators defined in [8]. $\mathbf{C}_*$ combinator has the property

(9) $\mathbf{C}_* af \equiv fa$

that is, it simply swaps the order of the functor and the argument. To obtain this equivalence, $\mathbf{C}_*$ must have the following λ-calculus definition: $\mathbf{C}_* \stackrel{\text{def}}{=} \lambda xf.fx$. Evaluating this combinator on only one argument (as done in type-raising; cf. the first derivation in 7) shows the equivalence of TR and $\mathbf{C}_*$ ($\geq$ means 'reduces to'):

$\lambda xf.fx(a) \geq \lambda f.fa$

Similarly, syntactic function composition corresponds to Curry's combinator **B**. **B** has the property

(10) $\mathbf{B}fgx \equiv f(gx)$

Thus $\mathbf{B} \stackrel{\text{def}}{=} \lambda fgx.f(gx)$. Evaluating it in the binary manner (as in 6) yields the semantics of these rules:

$\mathbf{B}fg \geq \lambda x.f(gx)$

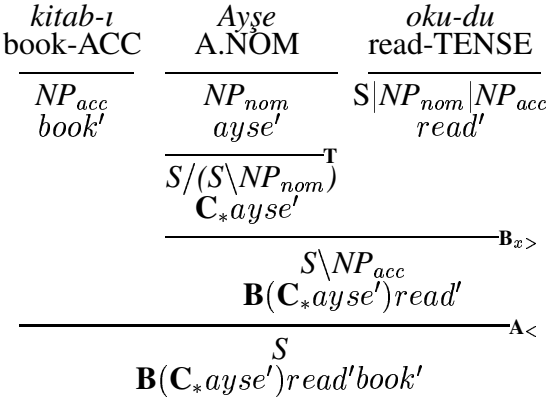
Some combinators, such as **W** (duplicator) and **K** (cancellator) have more of a lexical function, and do not operate at the syntactic level. For instance, **W** has the

definition $\mathbf{W} \stackrel{\text{def}}{=} \lambda f x.fxx$, which seems to suit the nature of reflexives (e.g., *John saw himself in the mirror*) [15]. Section 3 explains how the syntactic rules and their semantic counterparts can be modeled computationally to obtain semantic forms from surface structures.

3 Parsing into Combinator Expressions

Example (7) can now be written in combinator notation (11) instead of λ-calculus.

(11)



The final semantic form can be shown to be equivalent to that of (7):

$$\begin{aligned} \mathbf{B}(\mathbf{C}_* ayse')read'book' &\geq \mathbf{C}_* ayse'(read'book') && \text{by definition of } \mathbf{B} \\ &\geq read'book'ayse' && \text{by definition of } \mathbf{C}_* \end{aligned}$$

B can also reach in for the inner arguments of a verb with several arguments. For instance, $\mathbf{B}^2 = \mathbf{BB}$ is required for cases where the subject (i.e., the innermost argument) of a ditransitive verb scrambles to the left of the verb:

(12) *Mehmet'e* *kitab-ı* *Ayşe* *ver-di*
M.-DAT book-ACC A.NOM give-TENSE

NP_{dat} : $mehmet'$; NP_{acc} : $book'$; NP_{nom} : $ayse'$; $S|NP_{nom}|NP_{dat}|NP_{acc}$: $give'$

T: $S/(S\backslash NP_{nom})$: $\mathbf{C}_* ayse'$

$\mathbf{B}^2_{x>}$: $S|NP_{dat}|NP_{acc}$: $\mathbf{B}^2(\mathbf{C}_* ayse')give'$

$\mathbf{A}_<$: $S|NP_{dat}$: $\mathbf{B}^2(\mathbf{C}_* ayse')give'book'$

$\mathbf{A}_<$: S : $\mathbf{B}^2(\mathbf{C}_* ayse')give'book'mehmet'$

'Ayşe gave Mehmet the book.'

Reduction of the semantic representation into normal form shows that $\mathbf{B}^2$ is necessary to reduce on the correct argument of the verb; the innermost argument in syntactic category corresponds to the last argument in lambda abstraction of 3-place predicates like $give'$. The last line below reveals the predicate-argument structure of (12):

$$\begin{aligned} \mathbf{BB}(\mathbf{C}_* ayse')give'book'mehmet' &\geq \mathbf{B}(\mathbf{C}_* ayse')(give'book')mehmet' \\ &\geq \mathbf{C}_* ayse'(give'book'mehmet') \\ &\geq give'book'mehmet'ayse' \end{aligned}$$

(12) can be contrasted with (13). They have the same predicate-argument structure but *Ayşe* is the focus in (12), and *kitabı* in (13). Thus, they differ in information structure (topic, focus, background). The difference can be captured by having a topic-focus grammar interact with syntax, as proposed in [16]. The subject in (12) is more "marked"; it is in a position that would be associated with the default focus (the direct object) in the canonical word order SOV. This markedness is explicit in the syntactic derivations as well. The subject is the main functor and the verb is the secondary functor for $\mathbf{B}^2_{x>}$ in (12), whereas the verb is the main functor for $\mathbf{A}_<$ in (13). If one assigns intonational significance to primary-secondary functor distinction and to functor-argument distinction, the differences in the information structure can be revealed.

(13)

Ayşe	*Mehmet'e*	*kitab-ı*	*ver-di*
A.NOM	M.-DAT	book-ACC	give-TENSE
NP_{nom}	NP_{dat}	NP_{acc}	$S\|NP_{nom}\|NP_{dat}\|NP_{acc}$
$ayse'$	$mehmet'$	$book'$	$give'$
			$\mathbf{A}_<$
		$S\|NP_{nom}\|NP_{dat}$	
		$give'book'$	
			$\mathbf{A}_<$
		$S\|NP_{nom}$	
		$give'book'mehmet'$	
			$\mathbf{A}_<$
		S	
		$give'book'mehmet'ayse'$	

'Ayşe gave Mehmet the book.'

3.1 The Parsing Scheme

The parser expects as input the surface form of the utterance (as in the first row of 12) and produces a category and its semantic representation as output (as in the last two rows of 12). Reducing to a normal form, i.e., a form with no combinators, can be carried out by a post-processor of combinator expressions.

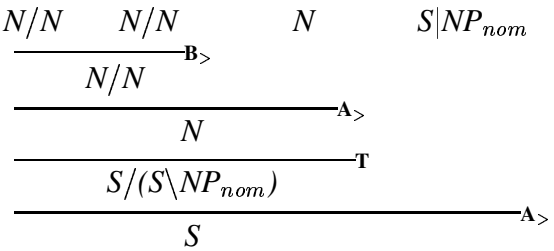
The parsing engine is a shift-reduce bottom-up parser that uses CG rules defined in (1), (5), and (6). For binary operations, the topmost two elements in the stack are matched against the left-hand sides of these rules. If a match is found, they are popped off, and the corresponding right-hand side is pushed onto the stack. Example below shows a sample derivation for *kitabı Ayşe okudu*:

Stack	Input	Parsing Action
\$	*kitabı Ayşe okudu*	shift
\$*kitabı*	*Ayşe okudu*	reduce
\$$NP_{acc}$		shift
\$$NP_{acc}$Ayşe	*okudu*	reduce
\$$NP_{acc}NP_{nom}$		reduce (by **T**)
\$$NP_{acc}(S/(S\backslash NP_{nom}))$		shift
\$$NP_{acc}(S/(S\backslash NP_{nom}))$okudu	-	reduce
\$$NP_{acc}(S/(S\backslash NP_{nom}))(S|NP_{nom}|NP_{acc})$	-	reduce (by $\mathbf{B}_{x>}$)
\$$NP_{acc}(S\backslash NP_{acc})$	-	reduce (by $\mathbf{A}_<$)
\$S	-	accept

Adding type-raising and composition to CG repertoire causes a problem known as 'spurious ambiguity' [17]: Two derivations are equivalent if they produce the same semantic form. Examples (14a) and (14b) are not genuinely ambiguous in this sense. A CG parser equipped with these two kinds of rules is called *structurally complete* [18], i.e., all well-formed syntactic bracketings of a surface string

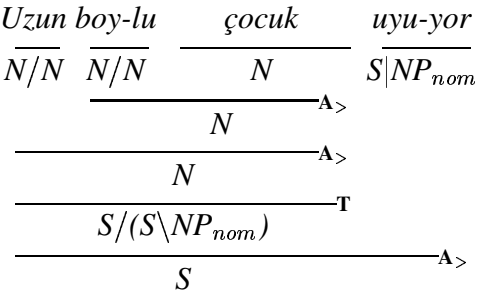
can be derived by such a parser. Although this is at first very alarming as far as parsing complexity is concerned, it is shown to be desirable in multi-grammar interaction, e.g., capturing the right intonational phrasing of a syntactic construction [16]. Moreover, it can be kept under control computationally[3] by parsing into normal forms [20, 21, 22], grammar re-writing [17], or by taking precautions in the parsing engine [23].

(14) a. *Uzun* *boy-lu* *çocuk* *uyu-yor*
long height-ADJ child.NOM sleep-TENSE



'The tall child is sleeping'

b. *Uzun* *boy-lu* *çocuk* *uyu-yor*



'The tall child is sleeping'

Syntax-semantics connection is accomplished by one-to-one pairing of syntactic rules with their equivalent combinator expressions. If a rule is applied to a syntactic form (i.e., categories), the semantics of that rule is applied to the semantic representation of the participants.

Proper bracketing of the combinators in the semantic form requires that their arity be specified in rules. For instance, although the combinator **B** by definition takes three elements (two functions and one argument), its use as a CG rule is binary; two functions are abstracted over an argument unknown at the time of rule application. We use the notation $X(A_1, \cdots, A_n)$ for a combinator X with arity

[3]CCG parsing can be done in polynomial time [19].

n. $\mathbf{C}_*$ has arity 1, and **B** has 2. We also assume that each semantic form in the lexicon specifies its own arity, e.g., 2 for transitive verbs. Similarly, $\mathbf{C}_*$ by definition has two arguments but it is used as a unary rule in CG. This is the essence of partial evaluation in syntax, which gives the added expressivity for deriving hitherto unanalyzable utterances. Using forward composition as an example, one-to-one coorespondence of syntax and semantics can be shown using the following convention,

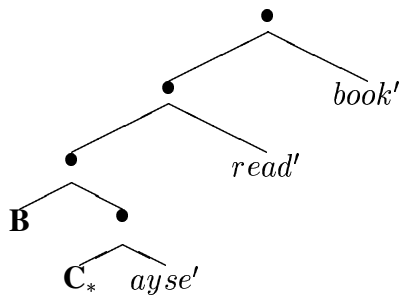
$$\begin{array}{lll} & \underline{\text{Syntax}} & \underline{\text{Semantics}} \\ X/Y \quad \overline{Y/Z} & \Rightarrow X/Z & f \;\; g \Rightarrow \mathbf{B}(\text{f,g}) \end{array}$$

3.2 Post-evaluation of Combinator Expressions

The semantic form obtained in parsing is a tree of combinator expressions. For $\mathbf{B}(\mathbf{C}_* ayse')read'book'$, the parser produces

A (**B** (**C**$_*$ ($ayse'$) , $read'$) , $book'$)

A represents function application (cf. 1). It is not a true combinator but the primitive of the combinator system. It stands for juxtaposition; we use $\bullet$ for the primitive in combinator trees. The tree for the output is as follows:



Tree construction can be performed in a bottom-up fashion: Figure 2 shows the local subtrees to be constructed for the binary primitive operation (a) and combinators (b).

With this representation, reduction into normal form amounts to the depth-first reduction of the combinator tree. This is akin to the graph reduction scheme employed in interpreting functional programming languages. Figure 1 shows some stages of such a reduction for $\mathbf{B}(\mathbf{C}_* child')give'book'ayse'$. Depth-first strategy selects the leftmost (outermost) combinator expression first, which corresponds

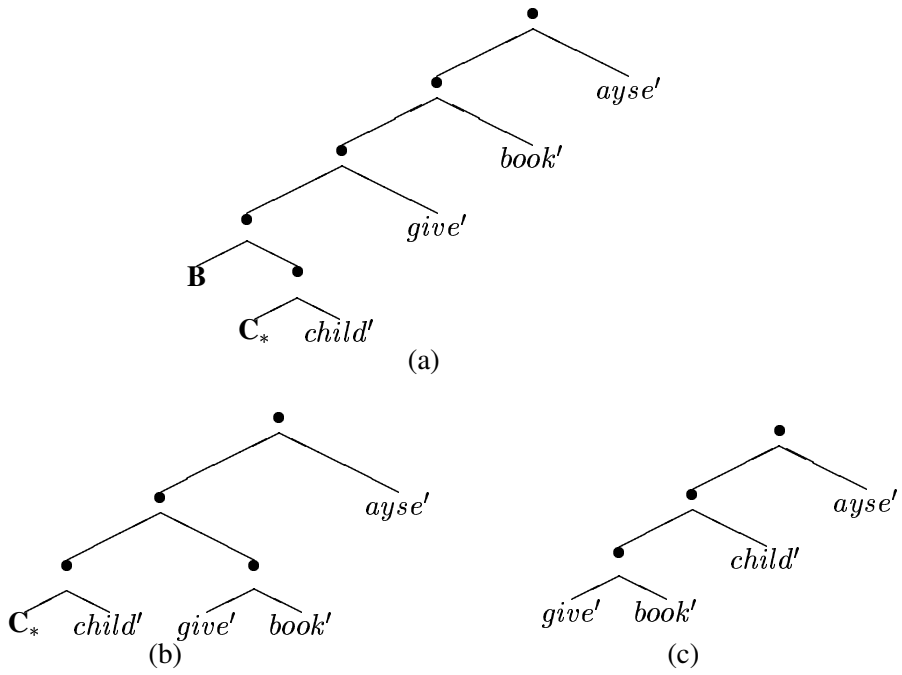
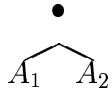
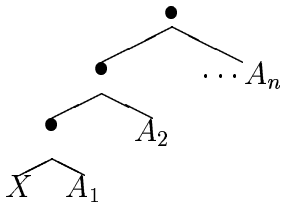


Figure 1: Stages of tree reduction for *Ayşe gave the child the book.*



Subtree for $\mathbf{A}(A_1, A_2)$

(a)



Subtree for $X(A_1, A_2, \cdots, A_n)$ where X is a combinator

(b)

Figure 2: Constructing subtrees for combinator expressions.

to *normal-order evaluation* [24]. According to the second Church-Rosser Theorem, if the semantic representation has a normal form at all, it can be obtained by normal-order evaluation. Decidability of this process depends on what set of combinators are made available in CG apparatus and the lexicon of a particular language. For instance, **W** is available in the lexicon as part of semantics of reflexive pronouns, and $\mathbf{C}_*$ may be lexical [25, 10] or syntactic [26] in English. It is both lexical and syntactic in Turkish [27]. If any arrangement of $\mathbf{W} \stackrel{\text{def}}{=} \lambda f x.fxx$ is allowed in lexicon or obtained in derivations, one might encounter **WWW**, which evaluates onto itself indefinitely:

$\mathbf{W}fx \equiv fxx$ thus $\mathbf{WWW} \geq \mathbf{WWW}$

Fortunately, such arrangements have not been encountered in natural language syntax, but it is an active philosophical debate as to whether natural languages are completely decidable.

$\mathbf{C}_*$ poses a different kind of parsing problem. If any arrangement of $\mathbf{C}_*$ is allowed, we might obtain expressions of type $\cdots \mathbf{C}_* X a \cdots$ where X is any combinator. The intermediate expression evaluates to aX. Since the functional symbol precedes the combinator, no other reduction may be possible if there is no other

combinator before a to carry on the evaluation. For this reason, combinators like $\mathbf{C}_*X$ are called *improper* by Curry, and $\mathbf{C}_*$ itself is considered an *irregular* combinator. Nevertheless, the controlled use of $\mathbf{C}_*$ is widespread in natural languages.

4 Conclusion

The primitives of Combinatory Logic can be shown to operate in natural language syntax. The correspondence is made explicit in the notation and methodology of Combinatory Categorial Grammars. Different syntactic realizations of these primitives, such as harmonic and disharmonic composition, type-raising in free-order (e.g., Turkish) vs. fixed-order languages (e.g., English) capture the typological differences in natural languages, but they also show the common mechanisms underlying these seemingly different operations.

We showed the manifestation of some of these operations in Turkish syntax. In particular, type-raising in Turkish accounts for scrambling, along with the crossing compositions. These operations have very restricted use (or none) in configurational languages. Compared to cumbersome movement rules and destructive tree operations in phrase structure accounts of word order variation, this approach offers a simple algebraic solution to the problem. The computational model built on these principles indicate how semantic composition in a parser can be performed in parallel with syntactic derivations. The semantic form obtained in this fashion can be interpreted very efficiently since it uses a logical notation that makes no use of bound variables hence do not carry any computational overhead for keeping track of bindings, scope, etc. Normal-order evaluation of resulting combinator expressions guarantees full interpretation of semantic representation, if there is one.

References

[1] Yehoshua Bar-Hillel. *Language and Information*. Addison-Wesley, Reading, Mass., 1964.

[2] Noam Chomsky. *Syntactic Structures*. Mouton, The Hague, 1957.

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

[4] Noam Chomsky. *The Minimalist Program*. MIT Press, Cambridge, Mass., 1995.

[5] Joachim Lambek. The mathematics of sentence structure. *American Mathematical Monthly*, 65:154–170, 1958.

[6] Richard Montague. The proper treatment of quantification in ordinary English. In Richard Thomason, editor, *Formal Philosphy: Selected Papers of Richard Montague*. Yale University Press, New Haven, CT, 1973.

[7] Moses Schönfinkel. On the building blocks of mathematical logic. In Jan van Heijenoort, editor, *From Frege to Gödel*. Harvard University Press, 1967, 1924.

[8] Haskell B. Curry and Robert Feys. *Combinatory Logic I*. North-Holland, Amsterdam, 1958.

[9] Mark Steedman. Dependency and coördination in the grammar of Dutch and English. *Language*, 61(3):523–568, 1985.

[10] Mark Steedman. Combinators and grammars. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*. D. Reidel, Dordrecht, 1988.

[11] Anna Szabolcsi. ECP in categorial grammar. ms., Max-Planck Institute, 1983.

[12] D. A. Turner. A new implementation technique for applicative languages. *Software—Practice and Experience*, 9:31–49, 1979.

[13] Stuart M. Shieber. *An Introduction to Unification-based Approaches to Grammar*. CSLI, Stanford, 1986.

[14] David Dowty. Type raising, functional composition, and non-constituent conjunction. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*. D. Reidel, Dordrecht, 1988.

[15] Anna Szabolcsi. Combinatory grammar and projection from the lexicon. In Ivan A. Sag and Anna Szabolcsi, editors, *Lexical Matters*. CSLI, Stanford, 1992.

[16] Mark Steedman. Structure and intonation. *Language*, 67:260–298, 1991.

[17] Kent Wittenburg. Predictive combinators. In *Proceedings of the 25th Annual Meeting of the ACL*, pages 73–79, 1989.

[18] Wojciech Buszkowski. Generative power of categorial grammars. In Richard T. Oehrle, Emmon Bach, and Deirdre Wheeler, editors, *Categorial Grammars and Natural Language Structures*. D. Reidel, Dordrecht, 1988.

[19] K. Vijay-Shanker and David J. Weir. Parsing some constrained grammar formalisms. *Computational Linguistics*, 19:591–636, 1993.

[20] Jason Eisner. Efficient normal-form parsing for combinatory categorial grammar. In *Proceedings of the 34th Annual Meeting of the ACL*, pages 79–86, 1996.

[21] Mark Hepple. Efficient incremental processing with categorial grammar. In *Proceedings of the 29th Annual Meeting of the ACL*, pages 79–86, 1991.

[22] Esther König. Parsing as natural deduction. In *Proceedings of the 27th Annual Meeting of the ACL*, pages 272–279, 1989.

[23] Remo Pareschi and Mark Steedman. A lazy way to chart-parse with categorial grammars. In *Proceedings of the 25th Annual Meeting of the ACL*, pages 81–88, 1987.

[24] Simon L. Peyton Jones. *The Implementation of Functional Programing Languages*. Prentice-Hall, New York, 1987.

[25] Lauri Karttunen. Radical lexicalism. In Mark Baltin and Anthony Kroch, editors, *Alternative Conceptions of Phrase Structure*. Chicago University Press, 1989.

[26] Mark Steedman. *Surface Structure and Interpretation*. MIT Press, Cambridge, Mass., 1997.

[27] Cem Bozsahin. Type raising, case marking, and directionality in Combinatory Categorial Grammar. Technical Report TR-96-2, Department of Computer Engineering, Middle East Technical University, 1996.

Bileşimsel Mantık ve Doğal Dil Ayrıştırması

Cem Bozşahin

Bilgisayar Mühendisliği Bölümü
Ortadoğu Teknik Üniversitesi
06531 Ankara

Özet

Bu çalışmada, Bileşimsel Mantık ile doğal dillerin sözdizimlerindeki işlemlerin ilişkileri irdelenmiştir. Türkçe'deki sözcük sırası değişiminin sözdizimsel özellikleri incelenmiş ve bu özellikler işlevsel birim elemanları ile tanımlanmıştır. Yüzey gösteriminden mantıksal gösterim türeten bir bilgisayarlarlı ayrıştırma modeli geliştirilmiştir. Bu işlemler, işlevsel programlama dillerin yorumlanmasında kullanılan yöntemlerle benzerlik göstermektedir.