Logic grammars and natural language

M.H. ter Beek

15 May 1997

Outline

- Basic features of logic grammars

  $\Rightarrow$ Describing natural language

  $\Rightarrow$ Describing formal languages

- Basic types of logic grammars and more enhanced features of natural language

  $\Rightarrow$ Metamorphosis Grammars (MGs)

  $\Rightarrow$ Definite Clause Grammars (DCGs)

  $\Rightarrow$ Extraposition Grammars (XGs)

  $\Rightarrow$ A comparison of the three types

- Further expressive power

  $\Rightarrow$ Discontinuous Grammars (DGs)

  $\Rightarrow$ Describing natural language

  $\Rightarrow$ Describing formal languages

  $\Rightarrow$ Static Discontinuity Grammars (SDGs)

  $\Rightarrow$ Government and Binding (GB) theory

  $\Rightarrow$ Grammar Systems & Natural Language

Basic features

Logic grammars look like *Chomsky grammars* in the formal language theory in that their rules have the form

$$\alpha \to \beta,$$

where α and β are strings of non-terminals and terminals, **BUT**

$\Rightarrow$ Grammar symbols may include arguments representing trees

$\Rightarrow$ Variables may be used, leading to unification

$\Rightarrow$ Tests may be included in a rule

Arguments have the general form

$$root(t_1, \ldots, t_n),$$

where *root* is an n-ary function symbol and the t_i, $1 \leq i \leq n$, are arguments representing trees, used recursively.

For example, the tree

noun-phrase

determiner noun

the boy

is represented in functional notation as

noun-phrase(determiner(the),noun(boy))

Variables, in turn, can be used to represent (sub-)trees whose values are hitherto unknown.

For example, the tree

$$\begin{array}{c}\text{question}\\ \text{F}\end{array} ,$$

containing the variable F, stands for all infinitely many trees that can be obtained by replacing F by some particular tree (which may again contain variables).

Rule application must involve more than just simple replacement. When necessary, assignment becomes the more general concept of *unification*.

Unification means that given two trees, those values are sought, that make the two trees identical. The set of these values is called a *substitution*.

For example, the trees

f and f

a X a b

unify with the substitution $\{X = b\}$. After unification, both trees have the form

f

a b

Unification can be used to specify tests within rules. Another way of specifying tests within rules is by *procedure calls* (in Prolog).

The right side of a logic grammar rule may contain procedure calls as well as grammar symbols.

A procedure call has the form

$$\{P_1, P_2, \ldots, P_n\} \text{ for } n \geq 1,$$

where the P_i, $1 \leq i \leq n$, are predicates for which Prolog definitions are given.

A *derivation* graph starts with the non-terminal called *start-symbol* and grows by application of rules. At any moment in time, given the string

$$\alpha\beta_1 \ldots \beta_n\gamma,$$

where the substring $\beta_1 \ldots \beta_n$ can be unified with the left side of a rule $\beta'_1 \ldots \beta'_n \rightarrow \delta_1 \ldots \delta_m$ with substitution θ, the application of this rule is drawn in the derivation graph as follows, where $\delta'_1 \ldots \delta'_m$ is $\delta_1 \ldots \delta_m$ affected by the substitution θ.

$$\begin{array}{c} \alpha\beta_1 \ldots \beta_n\gamma \\ \theta \\ \delta'_1 \ldots \delta'_m \end{array}$$

This process can be repeated for the new string $\alpha\delta'_1 \ldots \delta'_m\gamma$ until a string of terminals is obtained.

Describing natural language

The first idea for writing a logic grammar for describing simple sentences constructed from names, verbs and adjectives, could be

sentence $\rightarrow$ name, verb
sentence $\rightarrow$ name, verb, object
sentence $\rightarrow$ name, verb, place_complement
sentence $\rightarrow$ name, verb, time_complement
etc.

However, these rules are not only redundant, but also call for more redundant rules, since also the rules

place_complement $\rightarrow$ preposition, name
time_complement $\rightarrow$ preposition, name
etc.

need to be included in the grammar.

Generalization is necessary to simplify the grammar. A single symbol *comps* can be used to treat all complements of the verb, by stating for each verb its requirements concerning complements.

The goal in describing a given natural language by a grammar is to generalize as much as possible in order to use the fewest possible amount of rules, though not forgetting modularity. Not only does this lead to the most elegant grammar, but also the debugging is simplified and the possibility of errors minimized.

This is also the goal in linguistic research lately, which can be seen in the Government and Binding theory of Chomsky.

This grammar can handle easy Dutch sentences like "Tom is boos op Marie" (Tom is angry with Mary).

(S) sentence(S) $\rightarrow$ proper_noun(K), verb(K,L,S), complements(L)

(V) verb(K,L,S) $\rightarrow$ verb_1(be), adjective(K,L,S)

(A) adjective(K1,[arg(with,K2)],angry_with(K1,K2)) $\rightarrow$ adj_1(angry)

(C1) complements([]) $\rightarrow$ []

(C2) complements([arg(P,K) | L]) $\rightarrow$ complements(L), preposition(P), proper_noun(K)

The lexicon:

(L1) adj_1(angry) $\rightarrow$ [boos]

(L2) verb_1(be) $\rightarrow$ [is]

(L3) preposition(with) $\rightarrow$ [op]

(L4) proper_noun(tom) $\rightarrow$ [tom]

(L5) proper_noun(mary) $\rightarrow$ [marie]

Note that the convention (as in Prolog) to write *constants* in lowercase and to start *variables* with a capital, prohibits the proper names to start with a capital.

The derivation graph for " Tom is boos op Marie" .

Describing formal languages

Here are some different logic grammars for the formal language $\{a^n b^n c^n \mid n \geq 1\}$.

$$
\begin{aligned}
G_1: \quad & s \to [a] s [b] c \\
& s \to [\,] \\
& c[b] \to [b] c \\
& c \to [c]
\end{aligned}
$$

$$
\begin{aligned}
G_2: \quad & s \to a(n), b(n), c(n) \\
& a(1) \to [a] \\
& a(N) \to [a], \{M \text{ is } N-1\}, a(M) \\
& b(1) \to [b] \\
& b(N) \to [b], \{M \text{ is } N-1\}, b(M) \\
& c(1) \to [c] \\
& c(N) \to [c], \{M \text{ is } N-1\}, c(M)
\end{aligned}
$$

G_3:

$$
\begin{aligned}
& s \to sequence(n,a), sequence(n,b), sequence(n,c) \\
& sequence(1,X) \to [X] \\
& sequence(N,X) \to [X], \{M \text{ is } N-1\}, sequence(M,X)
\end{aligned}
$$

Cooperating distributed (CD) grammar systems:

(1) rules are grouped in components by listing them one by one

(2) a component is used by non-deterministically choosing one rule from it (and applying that rule) for as long as defined by its mode of derivation

(3) one mode of derivation for all components

Hybrid CD grammar systems: (1), (2) and

(4) components can have different modes of derivation

Prescribed hybrid team CD grammar systems:
(1) and

(5) components are grouped in teams by listing them one by one

(6) a team is used by non-deterministically choosing one rule from every component of the team (and applying those rules in parallel) for as long as defined by its mode of derivation

(7) teams can have different modes of derivation

The following prescribed hybrid team CD grammar system

$$G_4 = (\{A, B\}, \{a, b, c\}, S, P_1, P_2, P_3, P_4, P_5, \\ (Q_1, =1), (Q_2, =1), (Q_3, =1)),$$

where

$$\begin{aligned} P_1 &= \{S \to AB\}, \\ P_2 &= \{A \to aAb\}, \\ P_3 &= \{B \to cB\}, \\ P_4 &= \{A \to ab\}, \\ P_5 &= \{B \to c\}, \\ Q_1 &= \{P_1\}, \\ Q_2 &= \{P_2, P_3\} \text{ and} \\ Q_3 &= \{P_4, P_5\} \end{aligned}$$

also yields $\{a^n b^n c^n \mid n \geq 1\}$.

Basic types of logic grammars and...

All three logic grammars to be described next can describe the recursively enumerable class of languages of the Chomsky hierarchy.

For practical purposes, however, they are different.

...more enhanced features of natural language

These types of grammars shall be compared on how they describe natural language features as constituent movement and elision.

It will turn out that they can all do it very easily and concisely.

The running example is the sentence (actually noun-phrase)

The man that John saw laughed,

which can be seen as the surface expression of the more explicit deep structure form

The man [John saw the man] laughed,

where the second occurrence of "the man" has moved to the left and was subsumed into the relative pronoun "that".

Metamorphosis Grammars (MGs)

Their rules have the form

$$S\alpha \rightarrow \beta,$$

where S is a non-terminal grammar symbol, α is a string of non-terminals and terminals and β is a string of non-terminals, terminals and procedure calls.

The following grammar G_5 can treat the example sentence appropriately.

(S) sentence → noun_phrase, verb_phrase

(N1) noun_phrase → determiner, noun, relative

(N2) noun_phrase → proper_name

(V1) verb_phrase → verb

(V2) verb_phrase → transitive_verb, direct_object

(V3) verb_phrase → moved_dobj, transitive_verb

(R1) relative $\rightarrow$ []

(R2) relative $\rightarrow$ relative_marker, sentence

(RM) relative_marker, noun_phrase, moved_dobj $\rightarrow$

relative_pronoun, noun_phrase

(DO) direct_object $\rightarrow$ noun_phrase

(L1) determiner $\rightarrow$ [the]

(L2) noun $\rightarrow$ [man]

(L3) proper_name $\rightarrow$ [john]

(L4) verb $\rightarrow$ [laughed]

(L5) transitive_verb $\rightarrow$ [saw]

(L6) relative_pronoun $\rightarrow$ [that]

The MG derivation graph for "The man that John saw laughed".

Definite Clause Grammars (DCGs)

DCGs are a special case of MGs in that their rules are restricted to a single non-terminal symbol on the left side.

The reason for their introduction was ease of implementation and no loss in power. Since symbols may include arguments, also DCGs can describe the class of recursively enumerable languages (naturally, not as straightforward as MGs).

The following grammar G_6 can treat the example sentence appropriately. An argument is needed for *sentence*, *verb_phrase* and *direct_object* to prevent direct objects from being elided outside a relative clause, though.

This argument has the value "[]" if the direct object in the verb phrase of the sentence is not elided and the value "elided" if it is.

(S1) sentence $\rightarrow$ sent([])

(S2) sent(E) $\rightarrow$ noun_phrase, verb_phrase(E)

(N1) noun_phrase $\rightarrow$ determiner, noun, relative

(N2) noun_phrase $\rightarrow$ proper_name

(V1) verb_phrase([]) $\rightarrow$ verb

(V2) verb_phrase(E) $\rightarrow$ transitive_verb, direct_object(E)

(R1) relative $\rightarrow$ []

(R2) relative $\rightarrow$ relative_pronoun, sent(E)

(DO1) direct_object([]) $\rightarrow$ noun_phrase

(DO2) direct_object(elided) $\rightarrow$ []

(L1) determiner $\rightarrow$ [the]

(L2) noun $\rightarrow$ [man]

(L3) proper_name $\rightarrow$ [john]

(L4) verb $\rightarrow$ [laughed]

(L5) transitive_verb $\rightarrow$ [saw]

(L6) relative_pronoun $\rightarrow$ [that]

The DCG derivation graph for "The man that John saw laughed".

Extraposition Grammars (XGs)

XGs allow referring to unspecified strings in their rules, which is especially suitable for describing left extraposition of constituents.

Their rules have the form

$$\alpha_1 \ldots \alpha_2 \text{ etc. } \alpha_{n-1} \ldots \alpha_n \rightarrow \beta,$$

where the "..." specify skips (arbitrary string of grammar symbols), the α_i, $1 \leq i \leq n$, are strings of non-terminals and terminals and β is a string of non-terminals, terminals and procedure calls.

These rules work as follows. Any sequence of symbols of the form

$$\alpha_1 w_1 \alpha_2 w_2 \text{ etc. } \alpha_{n-1} w_{n-1} \alpha_n$$

with arbitrary w_i's, $1 \leq i \leq n-1$ can be rewritten into $\beta w_1 w_2$ etc. w_{n-1}. Thus the α_i's, $1 \leq i \leq n$, are rewritten into β and the intermediate skips w_i, $1 \leq i \leq n-1$, are rewritten sequentially to the right of β.

For example, for relative clauses the XG rule

$$\text{relative\_marker} \ldots \text{complement} \ \rightarrow \ [\text{that}]$$

allows the skipping of any intermediate substring that appears after a relative marker when searching for a complement and then to subsume both relative marker and complement into the relative pronoun "that", which is placed to the left of the skipped substring.

The following grammar G_6 can treat the example sentence appropriately.

(S) sentence → noun_phrase, verb_phrase

(N1) noun_phrase → determiner, noun, relative

(N2) noun_phrase → proper_name

(V1) verb_phrase → verb

(V2) verb_phrase → transitive_verb, direct_object

(R1) relative → []

(R2) relative → relative_marker, sentence

(RM) relative_marker...direct_object → relative_pronoun

(DO) direct_object → noun_phrase

(L1) determiner → [the]

(L2) noun → [man]

(L3) proper_name → [john]

(L4) verb → [laughed]

(L5) transitive_verb → [saw]

(L6) relative_pronoun → [that]

The XG derivation graph for "The man that John saw laughed".

A comparison of the three types

All three types are equivalent in power. They can all generate the class of recursively enumerable (type-0) languages of the Chomsky hierarchy.

⇒ MGs treat movement by actually moving the constituents around.

⇒ DCGs have to use extra arguments to carry all information relative to movements.

⇒ XGs can treat left extraposition by actually skipping intermediate substrings instead of moving the constituents that follow.

Which one to choose for a certain application is hence a matter of personal taste.

Discontinuous Grammars (DGs)

DGs are a generalization of XGs. However, they can be seen best as MGs with the added feature that unidentified strings of constituents can be referred to, to be repositioned copied or deleted at any position.

Moreover, they do not need to obey any nesting constraints. XGs require that two unidentified substrings are either totally independent or that one of them is completely included in the other.

More formally, their rules have the form

$$A, \alpha_0, skip(X_1), \alpha_1, \ldots, skip(X_n), \alpha_n \rightarrow \\ \beta_0, skip(X'_1), \beta_1, \ldots, skip(X'_m), \beta_m,$$

where A is a non-terminal, the α_i and β_i are strings of non-terminals and terminals and the $skip(X_i)$ and $skip(X'_i)$ are skip symbols different from the set of non-terminals and terminals, $0 \leq i \leq n$, $0 \leq i \leq m$.

For example, the DG rule

a,skip(X),b,skip(Y),c $\rightarrow$ skip(Y),c,b,skip(X)

applied to the string

a,e,f,b,d,c

with skips X=e,f and Y=d, yields

dcbef

Of course, the following rule has the same result.

a,e,f,b,d,c $\rightarrow$ d,c,b,e,f

However, if the rule is

a,skip(X),b,skip(Y),c $\rightarrow$ skip(X),c,b,skip(X)

the string becomes

efcbef

Note that DG rules are merely a shorthand notation for ordinary logic grammar rules. Their use is justified by collapsing several rules into one, suiting the goal of generality.

Moreover, the rules approach the way one analyses a sentence; focusing on the next relevant substring, leaving an intermediate one suspended in the back of one's mind, to be focused on only later, possibly repositioned with other more closely related substrings. So there is some psychological reality to this idea of skipping intermediate substrings.

Describing natural language

Consider the following grammar G_7, handling sentence coordination and the meaning representation of elided objects. It can handle the sentence "Mary saw and John heard the train".

(S1) sentence(and(S1,S2)) $\rightarrow$ sent(S1),and,sent(S2)

(S2) sent(S) $\rightarrow$ name(K),verb(K,P,S),object(P)

(O1) object(P) $\rightarrow$ determiner(X,P1,P),noun(X,P1)

(O2) object(P),and,skip(G),object(P) $\rightarrow$ [and],skip(G), object(P)

(D) determiner(X,P,the(X,P)) $\rightarrow$ [the]

(N) noun(X,train(X)) $\rightarrow$ [train]

(L1) verb(X,Y,saw(X,Y)) $\rightarrow$ [saw]

(L2) verb(X,Y,heard(X,Y)) $\rightarrow$ [heard]

(L3) name(mary) $\rightarrow$ [mary]

(L4) name(john) $\rightarrow$ [john]

The DG derivation graph for "Mary saw and John heard the train".

Describing formal languages

Here are some different DGs for the formal language $\{a^n b^m c^n d^m \mid m, n \geq 0\}$.

$$
\begin{aligned}
G_8 : \quad & s \to as, bs, cs, ds \\
& as \to [] \\
& as, skip(X), xc \to [a], as, skip(X) \\
& bs \to [] \\
& bs, skip(X), xd \to [b], bs, skip(X) \\
& cs \to [] \\
& cs \to xc, [c], cs \\
& ds \to [] \\
& ds \to xd, [d], ds
\end{aligned}
$$

$$
\begin{aligned}
G_9: \quad & s \to as, bs, cs, ds \\
& as, skip(G), cs \to [a], as, skip(G), [c], cs \\
& as, skip(G), cs \to skip(G) \\
& bs, skip(G), ds \to [b], bs, skip(G), [d], ds \\
& bs, skip(G), ds \to skip(G)
\end{aligned}
$$

Also the language $\{a^n b^n c^n \,|\, n \geq 1\}$ can be easily described by a DG, using the same technique.

$$
\begin{aligned}
G_{10}: \quad & s \to as, bs, cs \\
& as, skip(G1), bs, skip(G2), cs \to \\
& \quad [a], as, skip(G1), [b], bs, skip(G2), [c], cs \\
& as, skip(G1), bs, skip(G2), cs \to \\
& \qquad\qquad\qquad skip(G1), skip(G2)
\end{aligned}
$$

The following prescribed hybrid team CD grammar system

$$G_{11} = (\{A, B, C, D\}, \{a, b, c, d\}, S, P_1, \ldots, P_9,\\ (Q_1, {=}1), (Q_2, {=}1), (Q_3, {=}1), (Q_4, {=}1)),$$

where

$$\begin{aligned}
P_1 &= \{S \to ABCD\},\\
P_2 &= \{A \to aA\},\\
P_3 &= \{C \to cC\},\\
P_4 &= \{B \to bB\},\\
P_5 &= \{D \to dD\},\\
P_6 &= \{A \to a\},\\
P_7 &= \{B \to b\},\\
P_8 &= \{C \to c\},\\
P_9 &= \{D \to d\},\\
Q_1 &= \{P_1\},\\
Q_2 &= \{P_2, P_3\},\\
Q_3 &= \{P_4, P_5\} \text{ and}\\
Q_4 &= \{P_6, P_7, P_8, P_9\}
\end{aligned}$$

also yields $\{a^n b^m c^n d^m \mid m, n \geq 1\}$.

Static Dicontinuity Grammars (SDGs)

SDGs were introduced to handle enhanced natural language features such as Chomsky's Government and Binding (GB) theory.

Their rules have one of the following two forms.

(1) $\alpha \rightarrow \beta$,

(2) $\alpha_1 \rightarrow \beta_1, \alpha_2 \rightarrow \beta_2, \ldots, \alpha_n \rightarrow \beta_n$,

where α and α_i are non-terminals and β and β_i are sequences of non-terminals, terminals and procedure calls, $1 \leq i \leq n$.

Rules of type (2) can be used to rewrite strings of the form

$$\alpha'_1 w_1 \alpha'_2 w_2 \ldots \alpha'_n$$

into

$$(\beta_1 w_1 \beta_2 w_2 \ldots \beta_n)\tau,$$

where α unifies with α'_i with a most general unifier τ, the w_i are strings of non-terminals and terminals and no constraint prevents the production's application, $1 \leq i \leq n$.

Because the strings w_i, $1 \leq i \leq n$, do not move, the derivation can be seen as follows.

$$\ldots \alpha'_1 w_1 \alpha'_2 w_2 \ldots \alpha'_n \ldots$$

$$\beta_1\tau \quad \beta_2\tau \quad \ldots \beta_n\tau$$

Constraints have the form

constraint(Path,Node,Root):- body,

where "Path" is a path in a derivation, "Node" a node in that path under which no element can move out of a given zone, "Root" the root of that zone and "body" a list of Prolog calls.

Government and Binding (GB) theory

Only the principle of subjacency will be explained, it identifies the language-dependent concept of *bounding nodes*, which block forbidden movements in a language.

The movement rule in linguistics allows arbitrary movement into an empty position. Ungrammatical sentences generated by it are filtered out later. Subjacency states that the movement rule cannot cross more than one bounding node (as defined the particular language) <u>at a time</u>.

For example, the direct object in the sentence "John likes ___" embedded in the sentence

Who do you believe John likes?

was allowed to move to the head of the sentence in the form of the pronoun "who", by first crossing the *sentence* node of its immediate subsentence and then crossing the main *sentence* bounding node, like this:

$[_{S-bar}$ Who $[_S$ do you believe $[_{S-bar}$ ___ $[_S$ John likes ___]]]]

However, the ungrammatical sentence "Who do you wonder why John likes?" is disallowed by subjacency. The direct object would have to cross the two *sentence* nodes at once, since the position under *S-bar* that it moved to in the previous sentence is already occupied by "why", like this:

$*[_{S-bar}$ Who $[_S$ do you wonder $[_{S-bar}$ why $[_S$ John likes ___]]]]

Thus subjacency actually forbids a rule to move any constituent "X" to a position outside a domain "a" in this situation:

$$[_a \ldots [_b \ldots X \ldots] \ldots],$$

where both "a" and "b" are bounding nodes.

Constraints for SDGs are written as follows.

$$constraint([A \ldots [B \ldots] \ldots], B, A) :- \\ bounding(A), bounding(B).$$

In words, if a bounding node "A" has as descendant a bounding nod "B", then no descendant of "B" is allowed to move outside the domain with root "A".

As said before, subjacency is language-dependent, for example for English the bounding nodes are "sentence" and "noun-phrase". In SDGs subjacency is thus expressed in one very general rule, making it very easy to adapt the bounding nodes for other languages.