A fragment of English

E_0 is a small fragment of English consisting of very simple sentences, constructed with only intransitive and transitive (but no ditransitive) verbs, common nouns, proper names, pronouns and determiners. Typical sentences are:

A sheep drinks
Rachel herds the sheep
Jacob loves her

A fragment of English

Similar strings are not E_0- (and, hence, English-) sentences:

$*$*Rachel feed the sheep*

$*$*Rachel feeds herds the sheep*

$*$*The shepherds feeds the sheep*

$*$*Rachel feeds*

$*$*Jacob loves she*

$*$*Jacob loves Rachel the sheep*

$*$*Them herd the sheep*

A fragment of English

- All E_0 sentences have two components, a *subject*, realized as a noun phrase, and a *predicate*, realized as a verb phrase.
- A noun phrase can either be a proper name, such as Rachel, or a pronoun, such as they, or a common noun, possibly preceded by a determiner: the lamb or three sheep.
- A verb phrase consists of a verb, such as feed or sleeps, with a possible additional object, which is a noun phrase.

A fragment of English

Furthermore, there are constraints on the combination of phrases in E_0:

- The subject and the predicate must *agree* on number and person: if the subject is a third person singular, so must the verb be.
- Objects complement only – and all – the *transitive* verbs.
- When a pronoun is used, it is in the *nominative* case if it is in the subject position, and in the *accusative* case if it is an object.

A context-free grammar, G_0, for E_0

Example: A context-free grammar, G_0, for E_0

$$
\begin{array}{ccl}
S & \rightarrow & NP\ VP \\
VP & \rightarrow & V \\
VP & \rightarrow & V\ NP \\
NP & \rightarrow & D\ N \\
NP & \rightarrow & Pron \\
NP & \rightarrow & PropN \\
D & \rightarrow & \text{the, a, two, every, } \ldots \\
N & \rightarrow & \text{sheep, lamb, lambs, shepherd, water } \ldots \\
V & \rightarrow & \text{sleep, sleeps, love, loves, feed, feeds, herd, herds, } \ldots \\
Pron & \rightarrow & \text{I, me, you, he, him, she, her, it, we, us, they, them} \\
PropN & \rightarrow & \text{Rachel, Jacob, } \ldots
\end{array}
$$

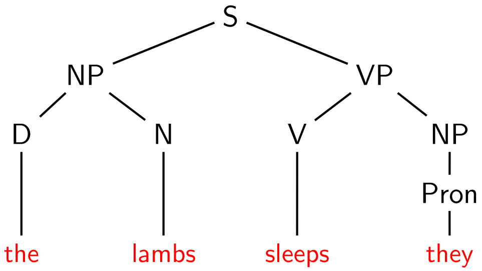
Problems of G_0

Over-generation (agreement constraints are not imposed):

$*$*Rachel feed the sheep*
$*$*The shepherds feeds the sheep*
$*$*Rachel feeds*
$*$*Jacob loves she*
$*$*Them herd the sheep*

Problems of G_0

Over-generation:



Problems of G_0

Over-generation (subcategorization constraints are not imposed):

the lambs sleep
Jacob loves Rachel
$*$*the lambs sleep the sheep*
$*$*Jacob loves*

Methodological properties of the CFG formalism

1. Concatenation is the only string combination operation
2. Phrase structure is the only syntactic relationship
3. The terminal symbols have no properties
4. Non-terminal symbols (grammar variables) are atomic
5. Most of the information encoded in a grammar lies in the production rules
6. Any attempt of extending the grammar with a semantics requires extra means.

Alternative methodological properties

1. Concatenation is not necessarily the only way by which phrases may be combined to yield other phrases.
2. Even if concatenation is the sole string operation, other syntactic relationships are being put forward.
3. Modern computational formalisms for expressing grammars adhere to an approach called *lexicalism*.
4. Some formalisms do not retain *any* context-free backbone. However, if one *is* present, its categories are not atomic.
5. The expressive power added to the formalisms allows also a certain way for representing semantic information.

Extending the CFG formalism

Formal issues:

- Motivation: imposing on a grammar for E_0 two of the restrictions violated by G_0: *person* and *number* agreement
- Introducing *feature structures*
- Extending the terminal symbols of a grammar
- Generalizing phrases and rules
- *Unification grammars*
- Imposing case control

Extending the CFG formalism

Linguistic issues:

- Subcategorization
- Feature structures for representing complex categories
- Subcategorization revisited
- Long-distance dependencies
- Subject/object control
- Coordination

Motivation

- The core idea is to incorporate into the grammar *properties* of symbols, in terms of which the violations of G_0 were stated.
- CFGs can be extended by associating feature structures with the terminal and non-terminal symbols of the grammar.
- To represent feature structures graphically we use a notation known as *attribute-value matrices* (AVMs).

Adding features to words

Words (terminal symbols) are endowed with structural information. The collection of enriched terminals is the grammar's *lexicon*.

Example: A lexicon

$$\overset{\textit{lamb}}{\begin{bmatrix}\text{NUM}: & \textit{sg}\\ \text{PERS}: & \textit{third}\end{bmatrix}} \quad \overset{\textit{lambs}}{\begin{bmatrix}\text{NUM}: & \textit{pl}\\ \text{PERS}: & \textit{third}\end{bmatrix}} \quad \overset{\textit{I}}{\begin{bmatrix}\text{NUM}: & \textit{sg}\\ \text{PERS}: & \textit{first}\end{bmatrix}}$$

$$\overset{\textit{sheep}}{\begin{bmatrix}\text{NUM}: & [\,]\\ \text{PERS}: & \textit{third}\end{bmatrix}} \quad \overset{\textit{dreams}}{\begin{bmatrix}\text{NUM}: & \textit{sg}\\ \text{PERS}: & \textit{third}\end{bmatrix}}$$

Complex values

Values can either be atomic, such as *sg*, or complex:

Example: A complex feature structure

$$\overset{\textit{lambs}}{\begin{bmatrix} \text{AGR}: & \begin{bmatrix} \text{NUM}: & \textit{pl} \\ \text{PERS}: & \textit{third} \end{bmatrix} \end{bmatrix}}$$

How to group features?

Variables

Example: Two notations for variables

$$\begin{bmatrix} \text{AGR}: & X\left(\begin{bmatrix} \text{NUM}: & \mathit{pl} \\ \text{PERS}: & \mathit{third} \end{bmatrix}\right) \end{bmatrix} \qquad \begin{bmatrix} \text{AGR}: & \boxed{1}\begin{bmatrix} \text{NUM}: & \mathit{pl} \\ \text{PERS}: & \mathit{third} \end{bmatrix} \end{bmatrix}$$

Diversion: feature structures

- Attribute-value matrices
- Equality and reentrancy
- Subsumption
- Unification
- Generalization
- Representing lists

Attribute-value matrices

- An AVM is a *syntactic* object, which can be either *atomic* or a *complex*.
- Each AVM is associated with a variable.
- An atomic feature structure is a variable, associated with an *atom*, drawn from a fixed set $\textsc{Atoms}$.
- A complex feature structure is a variable, associated with a finite, possibly empty set of pairs, where each pair consists of a *feature* and a *value*. Features are drawn from a fixed (per grammar), pre-defined set $\textsc{Feats}$; values are, recursively, AVMs themselves.

Attribute-value matrices

Example: AVM

$$\boxed{1}\begin{bmatrix}\textsc{agr}: & \boxed{2}\begin{bmatrix}\textsc{num}: & \boxed{3}\,\textit{pl}\\ \textsc{pers}: & \boxed{4}\,\textit{third}\end{bmatrix}\end{bmatrix}$$

- *pl*, *third* $\in$ ATOMS
- AGR, NUM, PERS $\in$ FEATS
- Thus, FEATS and ATOMS, as well as the (infinite) set of variables, are parameters for the collection of AVMs, and are referred to as the *signature*.
- We use meta-variables F, G, H to range over FEATS and A, B, C to range over feature structures.

Attribute-value matrices

Let

$$A = \boxed{i_0} \begin{bmatrix} \mathrm{F}_1 : & \boxed{i_1} A_1 \\ \vdots & \vdots \\ \mathrm{F}_n : & \boxed{i_n} A_n \end{bmatrix}$$

- $dom(A)$
- $[\,]$
- functionality: $\mathrm{F}_i \neq \mathrm{F}_j$
- $val(A, \mathrm{F}_i) = A_i$

Well-formed AVMs

Since variables are used to denote value sharing, there is not much sense in associating the same variable with two different values.

Example: Well-formed AVMs

$$A = Z\left(\begin{bmatrix} \text{F}: & X(a) \\ \text{G}: & Y\left(\begin{bmatrix} \text{H}: & X(a) \end{bmatrix}\right) \end{bmatrix}\right), B = \boxed{4}\begin{bmatrix} \text{F}: & \boxed{1}\begin{bmatrix} \text{H}: & \boxed{2}a \end{bmatrix} \\ \text{G}: & \boxed{1}b \end{bmatrix},$$

$$C = Z\left(\begin{bmatrix} \text{F}: & X\left(\begin{bmatrix} \text{H}: & Y(a) \end{bmatrix}\right) \\ \text{G}: & X\left(\begin{bmatrix} \text{K}: & W(b) \end{bmatrix}\right) \end{bmatrix}\right)$$

Conventions

- We assume in the sequel that AVMs are well-formed.
- Since multiple occurrences of the same variables always are associated with the same values, we usually only make explicit one instance of this value, leaving the other occurrences of the same variable unspecified.
- Whenever a variable is associated with the empty AVM we omit the AVM itself and leave the variable only.
- If a variable occurs only once in an AVM, we usually do not depict it (as it carries no additional information).

Conventions

Example: Shorthand notation for AVMs

Using our shorthand notation, the following AVM A :

$$A = Z\left(\begin{bmatrix} \text{F} : & X(a) \\ \text{G} : & Y\left(\begin{bmatrix} \text{H} : & X(a) \end{bmatrix}\right) \end{bmatrix}\right)$$

is depicted thus:

$$\begin{bmatrix} \text{F} : & X(a) \\ \text{G} : & \begin{bmatrix} \text{H} : & X \end{bmatrix} \end{bmatrix}$$

Conventions

Example: Conventions

A well-formed AVM:

$$\boxed{4}\begin{bmatrix} \text{F} : & \boxed{1}\,a \\ \text{G} : & \boxed{2}\begin{bmatrix} \text{F} : & \boxed{3}\,[\,] \\ \text{H} : & \boxed{1}\,a \end{bmatrix} \end{bmatrix}$$

Conventions

Example: Conventions

Removing multiple values of multiply-occurring variables:

$$\boxed{4}\begin{bmatrix} \mathrm{F}: & \boxed{1}\,a \\ \mathrm{G}: & \boxed{2}\begin{bmatrix} \mathrm{F}: & \boxed{3}\,[\,] \\ \mathrm{H}: & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Conventions

Example: Conventions

Removing the empty AVM:

$$\boxed{4}\begin{bmatrix} \mathrm{F}: & \boxed{1}\,a \\ \mathrm{G}: & \boxed{2}\begin{bmatrix} \mathrm{F}: & \boxed{3} \\ \mathrm{H}: & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Conventions

Example: Conventions

Removing non-informative variables:

$$\begin{bmatrix} \mathrm{F} : & \boxed{1}\,a \\ \mathrm{G} : & \begin{bmatrix} \mathrm{F} : & \boxed{3} \\ \mathrm{H} : & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Paths

A *path* is a (possibly empty) sequence of features that can be used to pick a value in a feature structure.
We use angular brackets '$\langle \ldots \rangle$' to depict paths explicitly.

Paths

Example: Paths

$$A = \begin{bmatrix} \text{F}: & X(a) \\ \text{G}: & \begin{bmatrix} \text{H}: & X \end{bmatrix} \end{bmatrix}$$

The single feature $\langle \text{F} \rangle$ constitutes a path; and so does the sequence $\langle \text{G},\text{H} \rangle$, since $\langle \text{G} \rangle$ can be used to pick the value $\begin{bmatrix} \text{H}: & X(a) \end{bmatrix}$ (because $val(A, \text{G}) = \begin{bmatrix} \text{H}: & X(a) \end{bmatrix}$).

Paths

- The notion of values is extended from features to paths: $val(A, \pi)$ is the value obtained by following the path π in A; this value (if defined) is again a feature structure.
- If A_i is the value of some path π in A then A_i is said to be a *sub-AVM* of A. The *empty path* is denoted ϵ, and $val(A, \epsilon) = A$ for every feature structure A.

Paths

Example: Basic notions

Let

$$A = \begin{bmatrix} \text{AGR} : & \begin{bmatrix} \text{NUM} : & \textit{pl} \\ \text{PERS} : & \textit{third} \end{bmatrix} \end{bmatrix}$$

Then

$$\textit{dom}(A) = \{\text{AGR}\}, \textit{val}(A, \text{AGR}) = \begin{bmatrix} \text{NUM} : & \textit{pl} \\ \text{PERS} : & \textit{third} \end{bmatrix}.$$

The paths of A are $\{\epsilon, \langle\text{AGR}\rangle, \langle\text{AGR,NUM}\rangle, \langle\text{AGR,PERS}\rangle\}$. The values of these paths are: $\textit{val}(A, \epsilon) = A$, $\textit{val}(A, \langle\text{AGR,NUM}\rangle) = \textit{pl}$, $\textit{val}(A, \langle\text{AGR,PERS}\rangle) = \textit{third}$. Since there is no path $\langle\text{NUM,AGR}\rangle$ in A, $\textit{val}(A, \langle\text{NUM,AGR}\rangle)$ is undefined.

Equality and reentrancy

When are two *atomic* feature structures equal?
Two atomic feature structures, whose variables are associated with one and the same atom, are not necessarily identical:

$$\begin{bmatrix} \mathrm{F}: & X(a) \\ \mathrm{G}: & Y(a) \end{bmatrix}$$

To ensure such identity, associate the same variable with the two values:

$$\begin{bmatrix} \mathrm{F}: & X(a) \\ \mathrm{G}: & X \end{bmatrix}$$

Equality and reentrancy

The features F and G are *reentrant*; a feature structure is reentrant if it contains (at least two) reentrant features.
To denote reentrancy in some feature structure A we use the symbol '$\overset{A}{\leftrightsquigarrow}$'.

Example: reentrancy

$$A_3 = \begin{bmatrix} \mathrm{F}_1 : & \begin{bmatrix} \mathrm{G} : & \boxed{1} \begin{bmatrix} \mathrm{H} : & b \end{bmatrix} \end{bmatrix} \\ \mathrm{F}_2 : & \begin{bmatrix} \mathrm{G} : & \boxed{1} \end{bmatrix} \end{bmatrix}$$

$\langle \mathrm{F}_1, \mathrm{G} \rangle \overset{A_3}{\leftrightsquigarrow} \langle \mathrm{F}_2, \mathrm{G} \rangle$.

Equality and reentrancy

Token identity vs. *type identity*
There is no path that can distinguish between the following two (non-identical!) structures:

$$A = \begin{bmatrix} \mathrm{F}: & a \\ \mathrm{G}: & a \end{bmatrix} \qquad B = \begin{bmatrix} \mathrm{F}: & \boxed{1}\, a \\ \mathrm{G}: & \boxed{1} \end{bmatrix}$$

Thus, feature structures are intensional objects.
When referring to type identity, we use the '$=$' symbol; to denote token identity we use the symbol '$\doteq$'.
$val(A, \pi_1) \doteq val(A, \pi_2)$ when $\pi_1 \overset{A}{\leftrightsquigarrow} \pi_2$.

Type-identity vs. token-identity

Example: Type-identity vs. token-identity

$$A = \begin{bmatrix} \text{SUBJ}: & \begin{bmatrix} \text{AGR}: & \begin{bmatrix} \text{NUM}: & \mathit{sg} \\ \text{PERS}: & \mathit{third} \end{bmatrix} \end{bmatrix} \\ \text{OBJ}: & \begin{bmatrix} \text{AGR}: & \begin{bmatrix} \text{NUM}: & \mathit{sg} \\ \text{PERS}: & \mathit{third} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$B = \begin{bmatrix} \text{SUBJ}: & \boxed{4} \begin{bmatrix} \text{AGR}: & \begin{bmatrix} \text{NUM}: & \mathit{sg} \\ \text{PERS}: & \mathit{third} \end{bmatrix} \end{bmatrix} \\ \text{OBJ}: & \boxed{4} \end{bmatrix}$$

Type-identity vs. token-identity

Example: Type-identity and token-identity revisited

Consider again the following feature structures:

$$A = \begin{bmatrix} \text{SUBJ}: & \begin{bmatrix} \text{AGR}: & \begin{bmatrix} \text{NUM}: & \mathit{sg} \\ \text{PERS}: & \mathit{third} \end{bmatrix} \end{bmatrix} \\ \text{OBJ}: & \begin{bmatrix} \text{AGR}: & \begin{bmatrix} \text{NUM}: & \mathit{pl} \\ \text{PERS}: & \mathit{third} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$$B = \begin{bmatrix} \text{SUBJ}: & \boxed{4} \begin{bmatrix} \text{AGR}: & \begin{bmatrix} \text{NUM}: & \mathit{sg} \\ \text{PERS}: & \mathit{third} \end{bmatrix} \end{bmatrix} \\ \text{OBJ}: & \boxed{4} \end{bmatrix}$$

$\mathit{val}(A, \text{SUBJ}) = \mathit{val}(A, \text{OBJ})$ and $\mathit{val}(B, \text{SUBJ}) = \mathit{val}(B, \text{OBJ})$. However, while $\mathit{val}(B, \text{SUBJ}) \doteq \mathit{val}(B, \text{OBJ})$, $\mathit{val}(A, \text{SUBJ}) \not\doteq \mathit{val}(A, \text{OBJ})$.

Example: (continued)

Suppose that by some action a feature CASE with the value *nom* is added to the value of SUBJ in both A and B:

$$A' = \begin{bmatrix} \text{SUBJ}: & \begin{bmatrix} \text{AGR}: & \begin{bmatrix} \text{NUM}: & sg \\ \text{PERS}: & third \end{bmatrix} \\ \text{CASE}: & nom \end{bmatrix} \\ \text{OBJ}: & \begin{bmatrix} \text{AGR}: & \begin{bmatrix} \text{NUM}: & pl \\ \text{PERS}: & third \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

$val(A', \text{SUBJ}) \neq val(A', \text{OBJ})$!

Example: (continued)

$$B' = \begin{bmatrix} \text{SUBJ}: & \boxed{4} \begin{bmatrix} \text{AGR}: & \begin{bmatrix} \text{NUM}: & \mathit{sg} \\ \text{PERS}: & \mathit{third} \end{bmatrix} \\ \text{CASE}: & \mathit{nom} \end{bmatrix} \\ \text{OBJ}: & \boxed{4} \end{bmatrix}$$

Here, $\mathit{val}(B', \text{SUBJ}) \doteq \mathit{val}(B', \text{OBJ})$, implying $\mathit{val}(B', \text{SUBJ}) = \mathit{val}(B', \text{OBJ})$.

Cycles

A special case of reentrancy is *cyclicity*: an AVM can contain a path whose value is the AVM itself. In other words, an AVM can be reentrant with a sub-structure of itself:

$$A = \begin{bmatrix} \text{F} : & \boxed{2} \begin{bmatrix} \text{G} : & a \\ \text{H} : & \boxed{2} \end{bmatrix} \end{bmatrix}$$

Renaming

If an AVM can be obtained from some other AVM through a systematic renaming of its variables, we say that each of the AVMs is a *renaming* of the other.

Example: Renaming

The following AVMs are renamings, as B can be obtained by renaming all the occurrences of the variable $\boxed{1}$ in A to $\boxed{4}$, and all the occurrences of $\boxed{2}$ to $\boxed{6}$:

$$A = \boxed{3}\begin{bmatrix} \mathrm{F}: & \boxed{2}\begin{bmatrix} \mathrm{G}: & \boxed{1}a \\ \mathrm{H}: & \boxed{2} \end{bmatrix} \\ \mathrm{G}: & \boxed{1} \end{bmatrix} \qquad B = \boxed{3}\begin{bmatrix} \mathrm{F}: & \boxed{6}\begin{bmatrix} \mathrm{G}: & \boxed{4}a \\ \mathrm{H}: & \boxed{6} \end{bmatrix} \\ \mathrm{G}: & \boxed{4} \end{bmatrix}$$