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

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

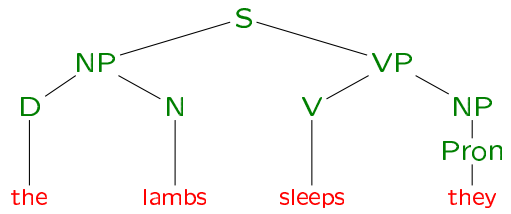
Problems of G_0

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

Overview

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

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

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

Complex values

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

Example: A complex feature structure

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

How to group features?

Variables

Example: Two notations for variables

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

Properties of 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 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 FEATS; values are, recursively, AVMs themselves.

Attribute-value matrices

$$\boxed{1}\begin{bmatrix}\text{AGR}: & \boxed{2}\begin{bmatrix}\text{NUM}: & \boxed{3}\ pl \\ \text{PERS}: & \boxed{4}\ third\end{bmatrix}\end{bmatrix}$$

- *pl*, *third* $\in$ ATOMS
- AGR, NUM, PERS $\in$ FEATS

Attribute-value matrices

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

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

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

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).

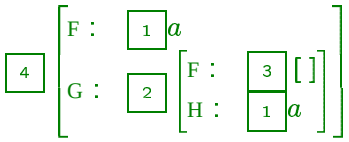
Conventions

Example: Shorthand notation for AVMs
Using our shorthand notation, the AVM A of example is depicted thus:

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

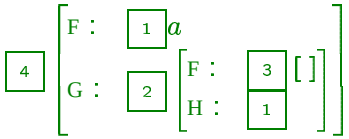
Conventions

A well-formed AVM:



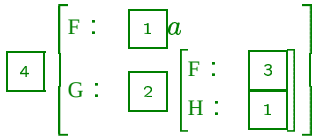
Conventions

Removing multiple values of multiply-occurring variables:



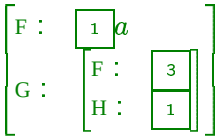
Conventions

Removing the empty AVM:



Conventions

Removing non-informative variables:



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

For example, in the AVM A of example , the single feature $\langle \text{F} \rangle$ constitutes a path; and so does the sequence $\langle \text{G,H} \rangle$, since $\langle \text{G} \rangle$ can be used to pick the value $\left[\text{H}: \quad X(a)\right]$ (because $val(A, \text{G}) = \left[\text{H}: \quad X(a)\right]$).

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

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

$$A = \left[\text{AGR}: \quad \begin{bmatrix} \text{NUM}: & pl \\ \text{PERS}: & third \end{bmatrix}\right]$$

Then

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

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: $val(A, \epsilon) = A$, $val(A, \langle \text{AGR,NUM} \rangle) = pl$, $val(A, \langle \text{AGR,PERS} \rangle) = third$. Since there is no path $\langle \text{NUM,AGR} \rangle$ in A, $val(A, \langle \text{NUM,AGR} \rangle)$ 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:

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

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

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

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 '$\overset{A}{\leftrightsquigarrow}$':

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

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

Equality and reentrancy

Token identity vs. *type identity*

There is no path that can distinguish between the following two (non-identical!) structures:

$$A = \begin{bmatrix} \text{F}: & a \\ \text{G}: & a \end{bmatrix} \quad B = \begin{bmatrix} \text{F}: & \boxed{1} a \\ \text{G}: & \boxed{1} \end{bmatrix}$$

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

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

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

Type-identity vs. token-identity

Example: Type-identity and token-identity revisited
Consider again the following feature structures:

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

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

$val(A, \text{SUBJ}) = val(A, \text{OBJ})$ and $val(B, \text{SUBJ}) = val(B, \text{OBJ})$. However, while $val(B, \text{SUBJ}) \doteq val(B, \text{OBJ})$, $val(A, \text{SUBJ}) \neq val(A, \text{OBJ})$.

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)

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

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

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 $\boxed{6}$:

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

This is a good time to stop.

Exercises?

Subsumption

Let A, B be feature structures over the same signature. We say that A subsumes B ($A \sqsubseteq B$; also, A is more general than B, and B is subsumed by, or is more specific than, A) if the following conditions hold:

1. if A is an atomic AVM then B is an atomic AVM with the same atom;
2. for every $\text{F} \in \textsc{Feats}$, if $\text{F} \in dom(A)$ then $\text{F} \in dom(B)$, and $val(A, \text{F})$ subsumes $val(B, \text{F})$; and
3. if two paths are reentrant in A, they are also reentrant in B: if $\pi_1 \overset{A}{\leftrightsquigarrow} \pi_2$ then $\pi_1 \overset{B}{\leftrightsquigarrow} \pi_2$.

Subsumption

Example: Subsumption

$$[\,] \sqsubseteq \begin{bmatrix} \text{NUM}: & sg \end{bmatrix}$$

$$\begin{bmatrix} \text{NUM}: & X \end{bmatrix} \sqsubseteq \begin{bmatrix} \text{NUM}: & sg \end{bmatrix}$$

$$\begin{bmatrix} \text{NUM}: & sg \end{bmatrix} \sqsubseteq \begin{bmatrix} \text{NUM}: & sg \\ \text{PERS}: & third \end{bmatrix}$$

$$\begin{bmatrix} \text{NUM1}: & sg \\ \text{NUM2}: & sg \end{bmatrix} \sqsubseteq \begin{bmatrix} \text{NUM1}: & \boxed{1}sg \\ \text{NUM2}: & \boxed{1} \end{bmatrix}$$

Subsumption

Subsumption is a partial relation: not every pair of feature structures is comparable:

$$\left[\text{NUM}: \; sg\right] \begin{array}{c} \not\sqsubseteq \\ \not\sqsupseteq \end{array} \left[\text{NUM}: \; pl\right]$$

A different case of incomparability is caused by the existence of different features in the two structures:

$$\left[\text{NUM}: \; sg\right] \begin{array}{c} \not\sqsubseteq \\ \not\sqsupseteq \end{array} \left[\text{PERS}: \; third\right]$$

Subsumption

Example: Subsumption
While subsumption informally encodes an order of information content among AVMs, sometimes the informal notion can be misleading:

$$\begin{bmatrix}\text{NUM}: & sg \\ \text{PERS}: & third\end{bmatrix} \begin{array}{c} \not\sqsubseteq \\ \not\sqsupseteq \end{array} \left[\text{AGR}: \; \begin{bmatrix}\text{NUM}: & sg \\ \text{PERS}: & third\end{bmatrix}\right]$$

Subsumption

Some properties of subsumption:

Least element: the empty feature structure subsumes every feature structure: for every feature structure A, $[\,] \sqsubseteq A$

Refelxivity: for every feature structure A, $A \sqsubseteq A$

Transitivity: If $A \sqsubseteq B$ and $B \sqsubseteq C$ than $A \sqsubseteq C$.

Antisymmetry: Subsumption is antisymmetric: if $A \sqsubseteq B$ and $B \sqsubseteq A$ then $A = B$.

To summarize, subsumption is a partial, reflexive, transitive and antisymmetric relation; it is therefore a *partial order*.

Unification

The unification operation, denoted '$\sqcup$', is defined over pairs of feature structures, and yields the most general feature structure that is more specific than both operands, if one exists:

$A = B \sqcup C$ if and only if A is the most general feature structure such that $B \sqsubseteq A$ and $C \sqsubseteq A$.

If such a structure exists, the unification succeeds, and the two arguments are said to be unifiable (or consistent).

If none exists, the unification fails, and the operands are said to be inconsistent.

Unification

Example: Unification

Unification combines consistent information:

$$\left[\text{NUM}: \quad sg\right] \sqcup \left[\text{PERS}: \quad third\right] = \begin{bmatrix}\text{NUM}: & sg \\ \text{PERS}: & third\end{bmatrix}$$

Different atoms are inconsistent: $\left[\text{NUM}: \quad sg\right] \sqcup \left[\text{NUM}: \quad pl\right] = \top$

Example: (continued)

Atoms and non-atoms are inconsistent: $\left[\text{NUM}: \quad sg\right] \sqcup sg = \top$

Example: (continued)

Unification is absorbing:

$$\left[\text{NUM}: \quad sg\right] \sqcup \begin{bmatrix}\text{NUM}: & sg \\ \text{PERS}: & third\end{bmatrix} = \begin{bmatrix}\text{NUM}: & sg \\ \text{PERS}: & third\end{bmatrix}$$

Example: (continued)

Empty feature structures are identity elements:

$$[\,] \sqcup \left[\text{AGR}: \quad \left[\text{NUM}: \quad sg\right]\right] = \left[\text{AGR}: \quad \left[\text{NUM}: \quad sg\right]\right]$$

Example: (continued)

Reentrancy causes two consistent values to coincide:

$$\begin{bmatrix} \text{F}: & \begin{bmatrix} \text{NUM}: & sg \end{bmatrix} \\ \text{G}: & \begin{bmatrix} \text{PERS}: & third \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{F}: & \boxed{1} \\ \text{G}: & \boxed{1} \end{bmatrix} = \begin{bmatrix} \text{F}: & \boxed{1} \begin{bmatrix} \text{NUM}: & sg \\ \text{PERS}: & third \end{bmatrix} \\ \text{G}: & \boxed{1} \end{bmatrix}$$

Example: (continued)

Variables can be (partially) instantiated:

$$\begin{bmatrix} \text{F}: & X \end{bmatrix} \sqcup \begin{bmatrix} \text{F}: & \begin{bmatrix} \text{H}: & b \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \text{F}: & X(\begin{bmatrix} \text{H}: & b \end{bmatrix}) \end{bmatrix}$$

Example: (continued)

Unification acts differently depending on whether the values are equal:

$$\begin{bmatrix} \text{F}: & \begin{bmatrix} \text{NUM}: & sg \end{bmatrix} \\ \text{G}: & \begin{bmatrix} \text{NUM}: & sg \end{bmatrix} \end{bmatrix} \sqcup \begin{bmatrix} \text{F}: & \begin{bmatrix} \text{PERS}: & third \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \text{F}: & \begin{bmatrix} \text{NUM}: & sg \\ \text{PERS}: & 3rd \end{bmatrix} \\ \text{G}: & \begin{bmatrix} \text{NUM}: & sg \end{bmatrix} \end{bmatrix}$$

...or identical:

$$\begin{bmatrix} \text{F}: & \boxed{1} \begin{bmatrix} \text{NUM}: & sg \end{bmatrix} \\ \text{G}: & \boxed{1} \end{bmatrix} \sqcup \begin{bmatrix} \text{F}: & \begin{bmatrix} \text{PERS}: & 3rd \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \text{F}: & \boxed{1} \begin{bmatrix} \text{NUM}: & sg \\ \text{PERS}: & 3rd \end{bmatrix} \\ \text{G}: & \boxed{1} \end{bmatrix}$$

Variable binding

Unification *binds* variables together. Let:

$$A = \begin{bmatrix} \text{F}: & \boxed{1} \begin{bmatrix} \text{NUM}: & sg \end{bmatrix} \end{bmatrix} \qquad B = \begin{bmatrix} \text{F}: & \boxed{2} \begin{bmatrix} \text{PERS}: & third \end{bmatrix} \end{bmatrix}$$

Then:

$$A \sqcup B = \begin{bmatrix} \text{F}: & \boxed{1}\boxed{2} \begin{bmatrix} \text{NUM}: & sg \\ \text{PERS}: & third \end{bmatrix} \end{bmatrix}$$

Of course, since the variables $\boxed{1}$ and $\boxed{2}$ occur nowhere else, they can be simply omitted and the result is equal to:

$$A \sqcup B = \begin{bmatrix} \text{F}: & \begin{bmatrix} \text{NUM}: & sg \\ \text{PERS}: & third \end{bmatrix} \end{bmatrix}$$

Variable binding

However, had either $\boxed{1}$ or $\boxed{2}$ occurred elsewhere (for example, as the value of some feature G in A), their values would have been modified as a result of the unification:

$$\begin{bmatrix} \text{F}: & \boxed{1}\begin{bmatrix}\text{NUM}: & sg\end{bmatrix} \\ \text{G}: & \boxed{1} \end{bmatrix} \sqcup \begin{bmatrix}\text{F}: & \boxed{2}\begin{bmatrix}\text{PERS}: & third\end{bmatrix}\end{bmatrix} =$$

$$\begin{bmatrix} \text{F}: & \boxed{3}\begin{bmatrix}\text{NUM}: & sg \\ \text{PERS}: & third\end{bmatrix} \\ \text{G}: & \boxed{3} \end{bmatrix}$$

Unification

Some properties of unification:

Idempotency: $A \sqcup A = A$

Commutativity: $A \sqcup B = B \sqcup A$

Associativity: $A \sqcup (B \sqcup C) = (A \sqcup B) \sqcup C$

Absorption: If $A \sqsubseteq B$ then $A \sqcup B = B$

Monotonicity: If $A \sqsubseteq B$ then for every C, $A \sqcup C \sqsubseteq B \sqcup C$ (if both exist).

Generalization

Generalization (denoted $\sqcap$) is the operation that returns the most specific (or least general) feature structure that is still more general than both arguments.

Unlike unification, generalization can never fail. For every pair of feature structures there exists a feature structure that is more general than both: in the most extreme case, pick the empty feature structure, which is more general than every other structure.

Generalization

Example: Generalization

Generalization reduces information:

$$\begin{bmatrix}\text{NUM}: & sg\end{bmatrix} \sqcap \begin{bmatrix}\text{PERS}: & third\end{bmatrix} = [\,]$$

Different atoms are inconsistent:

$$\begin{bmatrix}\text{NUM}: & sg\end{bmatrix} \sqcap \begin{bmatrix}\text{NUM}: & pl\end{bmatrix} = \begin{bmatrix}\text{NUM}: & [\,]\end{bmatrix}$$

Example: (continued)

Generalization is restricting:

$$\begin{bmatrix}\text{NUM}: & sg\end{bmatrix} \sqcap \begin{bmatrix}\text{NUM}: & sg \\ \text{PERS}: & third\end{bmatrix} = \begin{bmatrix}\text{NUM}: & sg\end{bmatrix}$$

Example: (continued)

Empty feature structures are zero elements:

$$[\,] \sqcap \begin{bmatrix}\text{AGR}: & \begin{bmatrix}\text{NUM}: & sg\end{bmatrix}\end{bmatrix} = [\,]$$

Reentrancies can be lost:

$$\begin{bmatrix}\text{F}: & \boxed{1}\ \begin{bmatrix}\text{NUM}: & sg\end{bmatrix} \\ \text{G}: & \boxed{1}\end{bmatrix} \sqcap \begin{bmatrix}\text{F}: & \begin{bmatrix}\text{NUM}: & sg\end{bmatrix} \\ \text{G}: & \begin{bmatrix}\text{NUM}: & sg\end{bmatrix}\end{bmatrix} = \begin{bmatrix}\text{F}: & \begin{bmatrix}\text{NUM}: & sg\end{bmatrix} \\ \text{G}: & \begin{bmatrix}\text{NUM}: & sg\end{bmatrix}\end{bmatrix}$$

Generalization

Some properties of generalization:

Idempotency: $A \sqcap A = A$

Commutativity: $A \sqcap B = B \sqcap A$

Absorption: If $A \sqsubseteq B$ then $A \sqcap B = A$

Using feature structures for representing lists

Feature structures can be easily used to encode (finite) lists. As an example, consider the following representation of the list $\langle 1,2,3\rangle$ (assuming a signature whose atoms include the numbers 1,2,3):

Example: Feature structure encoding of a list

$$\begin{bmatrix}\text{FIRST}: & 1 \\ \text{REST}: & \begin{bmatrix}\text{FIRST}: & 2 \\ \text{REST}: & \begin{bmatrix}\text{FIRST}: & 3 \\ \text{REST}: & elist\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Using feature structures for representing lists

Example: A nested list
The list $\langle\langle 1,2,6,7\rangle,\langle 3,4\rangle,\langle 5\rangle\rangle$:

$$\begin{bmatrix}
\text{FIRST}: & \begin{bmatrix}
\text{FIRST}: & 1 \\
\text{REST}: & \begin{bmatrix}
\text{FIRST}: & 2 \\
\text{REST}: & \begin{bmatrix}
\text{FIRST}: & 6 \\
\text{REST}: & \begin{bmatrix}\text{FIRST}: & 7 \\ \text{REST}: & \textit{elist}\end{bmatrix}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix} \\
\text{REST}: & \begin{bmatrix}
\text{FIRST}: & \begin{bmatrix}
\text{FIRST}: & 3 \\
\text{REST}: & \begin{bmatrix}\text{FIRST}: & 4 \\ \text{REST}: & \textit{elist}\end{bmatrix}
\end{bmatrix} \\
\text{REST}: & \begin{bmatrix}
\text{FIRST}: & \begin{bmatrix}\text{FIRST}: & 5 \\ \text{REST}: & \textit{elist}\end{bmatrix} \\
\text{REST}: & \textit{elist}
\end{bmatrix}
\end{bmatrix}
\end{bmatrix}$$

Adding features to rules

Phrases, like words, have valued features and consequently, grammar non-terminals, too, are decorated with features.

When a feature is assigned to a non-terminal symbol C, it means that this feature is appropriate for *all* the phrases of category C: it makes sense for it to appear in all the instances of C.

Such categories interact, in the grammar, with other AVMs, through application of rules, and the specified values might thus undergo changes. In general, AVMs are changed as a result of rule application.

We refer to such enriched categories as *generalized categories* (or *extended* ones), which have a base category and an associated feature structure.

Adding features to rules

Example:
A feature structure that might be associated with phrases of category *NP* (noun phrases).

$$\begin{matrix}\textit{NP}\\ \begin{bmatrix}\text{NUM}: & [\,] \\ \text{PERS}: & [\,]\end{bmatrix}\end{matrix}$$

A third person singular noun phrase such as lamb may be associated with:

$$\begin{matrix}\textit{NP}\\ \begin{bmatrix}\text{NUM}: & \textit{sg} \\ \text{PERS}: & \textit{third}\end{bmatrix}\end{matrix}$$

Extended grammar rules

Example: Rules for imposing number agreement

(1) $\begin{matrix}N\\ [\text{NUM}: \; X]\end{matrix} \rightarrow \begin{matrix}\textit{lamb}\\ [\text{NUM}: \; X(\textit{sg})]\end{matrix}$

(2) $\begin{matrix}N\\ [\text{NUM}: \; X]\end{matrix} \rightarrow \begin{matrix}\textit{lambs}\\ [\text{NUM}: \; X(\textit{pl})]\end{matrix}$

(3) $\textit{S} \rightarrow \begin{matrix}\textit{NP} & \textit{VP}\\ [\text{NUM}: \; X] & [\text{NUM}: \; X]\end{matrix}$

(4) $\begin{matrix}\textit{NP}\\ [\text{NUM}: \; X]\end{matrix} \rightarrow \begin{matrix}\textit{D} & \textit{N}\\ [\text{NUM}: \; X] & [\text{NUM}: \; X]\end{matrix}$

Scope of variables

The scope of a variable is the grammar rule in which it occurs. Reformulating rule (2) as

$$\underset{\left[\text{NUM}: \quad Y\right]}{N} \rightarrow \underset{\left[\text{NUM}: \quad Y(pl)\right]}{lambs}$$

has no effect.

No sharing is implied by occurrences of the same variables in different rules, for example the occurrences of X in rules (1) and (2) above.

Declarativity

Rule (4) stipulates that in order to form a noun phrase (NP) from the concatenation of a determiner (D) and a noun (N), the NUM features of the determiner and the noun must agree.

The NUM feature of the noun phrase thus constructed is equal to that of either daughter.

What the rule does not determine is an order of this value check.

The undirectional view of agreement is a typical view of unification-based grammar formalisms.

Extending AVMs

Multi-AVMs can be viewed as sequences of AVMs, with the important observation that some sub-structures can be shared among two or more AVMSs.

In other words, the scope of variables is extended from a single AVM to a multi-AVM: the same variable can be associated with two sub-structures of different AVMs.

The notion of well-formedness is extended to multi-AVMs.

The function val, associating a value with features (and paths) has to be extended, too.

If the value of the path π_1 leaving the i-th root of σ is reentrant with the value of the path π_2 leaving the j-th root, we write $(i, \pi_1) \overset{\sigma}{\leftrightsquigarrow} (j, \pi_2)$.

Extending AVMs

Example: Multi-AVM

Let σ be the multi-AVM: $\left[\text{F}: \left[\begin{matrix}\text{G}: & a\\ \text{H}: & X\end{matrix}\right]\right] \quad \left[\text{G}: \quad Y\right] \quad \left[\begin{matrix}\text{F}: & \left[\begin{matrix}\text{H}: & b\\ \text{G}: & X\end{matrix}\right]\\ \text{H}: & a\end{matrix}\right]$

Then $val(\sigma, 1, \langle \text{F} \rangle)$ is: $\left[\begin{matrix}\text{G}: & a\\ \text{H}: & [\,]\end{matrix}\right]$

whereas $val(\sigma, 3, \langle \text{F} \rangle)$ is: $\left[\begin{matrix}\text{H}: & b\\ \text{G}: & [\,]\end{matrix}\right]$

In this example, $(1, \langle \text{F H} \rangle) \overset{\sigma}{\leftrightsquigarrow} (3, \langle \text{F G} \rangle)$.

Example: (continued)
A multi-AVM can have an empty feature structure as an element:

$$\left[\text{F}: \left[\begin{matrix}\text{G}: & a\\ \text{H}: & X\end{matrix}\right]\right] \quad [\,] \quad \left[\begin{matrix}\text{F}: & \left[\begin{matrix}\text{H}: & b\\ \text{G}: & X\end{matrix}\right]\\ \text{H}: & a\end{matrix}\right]$$

Extending AVMs

The following is a valid multi-AVM:

$$\left[\begin{matrix}\text{F}: & a\\ \text{G}: & \boxed{2}\end{matrix}\right] \quad \boxed{2}\left[\text{H}: \quad b\right]$$

The only restriction is that the same variable cannot be associated with two different elements in the sequence. Thus, the following is not a multi-AVM:

$$\boxed{2}\left[\text{H}: \quad b\right] \quad \left[\begin{matrix}\text{F}: & a\\ \text{G}: & \boxed{2}\end{matrix}\right] \quad \boxed{2}$$

Subsumption

The notion of subsumption can be naturally extended from AVMs to multi-AVMs: if σ and ρ are two multi-AVMs *of the same length*, n, then $\sigma \sqsubseteq \rho$ if the following conditions hold:

1. every element of σ subsumes the corresponding element of ρ: for every i, $1 \leq i \leq n$, $val(\sigma, i, \epsilon) \sqsubseteq val(\rho, i, \epsilon)$; and

2. if two paths are reentrant in σ, they are also reentrant in ρ: if $(i, \pi_1) \overset{\sigma}{\leftrightsquigarrow} (j, \pi_2)$ then $(i, \pi_1) \overset{\rho}{\leftrightsquigarrow} (j, \pi_2)$.

Subsumption

Example: Multi-AVM subsumption

Let σ be: $\left[\text{F}: \left[\begin{matrix}\text{G}: & a\\ \text{H}: & X\end{matrix}\right]\right] \quad \left[\text{G}: \quad c\right] \quad \left[\begin{matrix}\text{F}: & \left[\begin{matrix}\text{H}: & b\\ \text{G}: & X(d)\end{matrix}\right]\\ \text{H}: & a\end{matrix}\right]$

and ρ be: $\left[\text{F}: \left[\begin{matrix}\text{G}: & a\\ \text{H}: & d\end{matrix}\right]\right] \quad \left[\text{G}: \quad c\right] \quad \left[\begin{matrix}\text{F}: & \left[\begin{matrix}\text{H}: & b\\ \text{G}: & d\end{matrix}\right]\\ \text{H}: & a\end{matrix}\right]$

Then σ does not subsume ρ, but $\rho \sqsubseteq \sigma$.

Unification

In the same way, the notion of unification can be extended to multi-AVMs (of the same length): we say that ρ is the unification of σ_1 and σ_2 (and write $\rho = \sigma_1 \sqcup \sigma_2$) if σ_1, σ_2 and ρ are of the same length, and ρ is the most general multi-AVM that is more specific than both σ_1 and σ_2.

Rules and grammars

An extended context-free rule consists of two components: a context-free rule, and a multi-AVM of the same length.

A unification grammar consists of a set of extended context-free rules and an extended category that serves as the *start symbol*.

Unification grammars

Example: G_1, a unification grammar for E_0

(1) $S \rightarrow \underset{[\text{NUM}: X]}{NP} \quad \underset{[\text{NUM}: X]}{VP}$

(2) $\underset{[\text{NUM}: X]}{NP} \rightarrow \underset{[\text{NUM}: X]}{D} \quad \underset{[\text{NUM}: X]}{N}$

(3) $\underset{[\text{NUM}: X]}{VP} \rightarrow \underset{[\text{NUM}: X]}{V}$

Example: (continued)

(4) $\underset{[\text{NUM}: X]}{VP} \rightarrow \underset{[\text{NUM}: X]}{V} \quad \underset{[\text{NUM}: Y]}{NP}$

(5,6) $\underset{[\text{NUM}: X]}{N} \rightarrow \underset{[\text{NUM}: X(sg)]}{\textit{lamb}} \mid \underset{[\text{NUM}: X]}{\textit{sheep}} \mid \ldots$

(7,8) $\underset{[\text{NUM}: X]}{V} \rightarrow \underset{[\text{NUM}: X(sg)]}{\textit{sleeps}} \mid \underset{[\text{NUM}: X(pl)]}{\textit{sleep}} \mid \ldots$

(9,10) $\underset{[\text{NUM}: X]}{D} \rightarrow \underset{[\text{NUM}: X(sg)]}{\textit{a}} \mid \underset{[\text{NUM}: X(pl)]}{\textit{two}} \mid \ldots$

Rule application

- Forms and sentential forms
- Derivations
- Derivation trees
- Language

Forms

Forms are generalized and are composed of "sequences" of generalized categories, that is, of a sequence of base categories or words, augmented by a multi-AVM of the same length.

We use Greek letters such as α, β as meta-variables over forms. For example, following is a form of length two:

$$\begin{array}{cc} NP & VP \\ [\textsc{num}: \ Y] & [\textsc{num}: \ Y] \end{array}$$

Derivations

Derivation is a binary relation over generalized forms. Let α be a generalized form, and $B_0 \to B_1 B_2 \ldots B_k$ a grammar rule, where the B_i are all generalized categories, and where reentrancies might occur among elements of the form or the rule. Application of the rule to α consists of the following steps:

- Matching the rule's head with some element of the form that has the same base category;
- Replacing the selected element in the form with the body of the rule, producing a new form.

Derivations

Example: Matching
Suppose that

$$\alpha = \begin{array}{cc} NP & VP \\ [\textsc{num}: \ Y] & [\textsc{num}: \ Y] \end{array}$$

is a (sentential) form and that

$$\rho = \begin{array}{ccc} VP & & V \\ [\textsc{num}: \ X] & \to & [\textsc{num}: \ X] \end{array} \quad \begin{array}{c} NP \\ [\textsc{num}: \ W] \end{array}$$

is a rule. Let the selected element of α be its second element, namely the extended category

$$\begin{array}{c} VP \\ [\textsc{num}: \ Y] \end{array}$$

Example: (continued)
This extended category matches the head of the rule ρ, as the base categories are identical (*VP*) and the AVMs associated with them are unifiable (consistent). The result of the unification is the extended category

$$\overset{VP}{\left[\text{NUM}: \quad Z\right]}$$

which is equivalent to

$$\overset{VP}{\left[\text{NUM}: \quad [\,]\right]}$$

An additional effect of the unification is that the variables Y of the form and X of the rule are unified, too.

Replacement

The two feature structures (associated with the head of the rule and with the selected element) are unified in their respective contexts: the body of the rule and the form.

When some variable X in the form is unified with some variable Y in the rule, all occurrences of X in the form and of Y in the rule are modified: they are all set to the unified value.

The replacement operation inserts the modified rule body into the modified form, replacing the selected element of the form.

The variables of the resulting form are then systematically renamed.

Derivation

Example: Derivation step
Let

$$\alpha = \overset{NP}{\left[\text{NUM}: \quad Y\right]} \quad \overset{VP}{\left[\text{NUM}: \quad Y\right]}$$

$$\rho = \overset{VP}{\left[\text{NUM}: \quad X\right]} \rightarrow \overset{V}{\left[\text{NUM}: \quad X\right]} \quad \overset{NP}{\left[\text{NUM}: \quad W\right]}$$

be a form and a rule, respectively. The unification of the rule's head with the second element of α succeeds, and identifies the values of X and Y . After replacement and variable renaming we obtain:

$$\beta = \overset{NP}{\left[\text{NUM}: \quad X_1\right]} \quad \overset{V}{\left[\text{NUM}: \quad X_1\right]} \quad \overset{NP}{\left[\text{NUM}: \quad W_1\right]}$$

Example: (continued)
Now assume that the (terminal) rule

$$\overset{V}{\left[\text{NUM}: \quad Y\right]} \rightarrow \overset{herds}{\left[\text{NUM}: \quad Y(sg)\right]}$$

is to be applied to β. The value of the variable X_1 in the form is set, through unification, to sg, and the resulting form is:

$$\gamma = \overset{NP}{\left[\text{NUM}: \quad X_2\right]} \quad \overset{herds}{\left[\text{NUM}: \quad X_2(sg)\right]} \quad \overset{NP}{\left[\text{NUM}: \quad W_2\right]}$$

Note that the first NP had its feature structure modified, even though it did not participate directly in the rule application.

Derivation

Example: Derivation step (continued)
Assume now that γ is expanded by applying to its first element the rule

$$\overset{NP}{[\text{NUM}: X]} \rightarrow \overset{D}{[\text{NUM}: X]} \quad \overset{N}{[\text{NUM}: X]}$$

In this case, unification of the first element of γ with the head of the rule binds the value of X in the rule to *sg*:

$$\delta = \overset{D}{[\text{NUM}: X_3]} \quad \overset{N}{[\text{NUM}: X_3]} \quad \overset{herds}{[\text{NUM}: X_3(sg)]} \quad \overset{NP}{[\text{NUM}: W_3]}$$

Example: (continued)
If we now tried to apply the (terminal) rule

$$\overset{D}{[\text{NUM}: Y]} \rightarrow \overset{two}{[\text{NUM}: Y(pl)]}$$

to the first element of δ, this attempt would have caused unification failure.

Derivation

Example: Derivation with ϵ-rules
Let

$$\alpha = \overset{A}{[\text{F}: X]} \quad \overset{B}{\begin{bmatrix}\text{F}: X\\ \text{G}: Y\end{bmatrix}} \quad \overset{C}{[\text{G}: Y]}, \quad \rho = \overset{B}{\begin{bmatrix}\text{F}: Z\\ \text{G}: Z\end{bmatrix}} \rightarrow \epsilon$$

be a form and a rule, respectively. Applying the rule to the second element of the form yields:

$$\overset{A}{[\text{F}: W]} \quad \overset{C}{[\text{G}: W]}$$

Derivation

Example: Derivation can modify information
Let

$$\alpha = \overset{A}{[\text{F}: a]} \quad \overset{B}{[\text{G}: b]}, \quad \rho = \overset{A}{[\text{F}: a]} \rightarrow \overset{A}{[\text{F}: c]}$$

be a form and a rule, respectively. Applying the rule to the first element of the form yields:

$$\overset{A}{[\text{F}: c]} \quad \overset{B}{[\text{G}: b]}$$

Notice that in the result, the value of F in *A* was modified from a to c.

Derivation

The full derivation relation is, as usual, the reflexive-transitive closure of rule application.

A form is *sentential* if it is derivable from the start symbol.

Derivation

A derivation of the sentence two sheep sleep with the grammar G_1. After each rule is applied, the variables in the obtained form are renamed.

Example: Derivation
The derivation starts with the start symbol, which is the extended category S. Applying rule (1), one gets:

$$\underset{NP}{[\text{NUM}: X_1]} \quad \underset{VP}{[\text{NUM}: X_1]}$$

Example: (continued)
It is now possible to select the leftmost element in the above sentential form and to apply rule (2). Renaming all occurrences of X in rule (2) to X_2, one gets the following sentential form:

$$\underset{D}{[\text{NUM}: X_2]} \quad \underset{N}{[\text{NUM}: X_2]} \quad \underset{VP}{[\text{NUM}: X_2]}$$

Now select the rightmost element in the above form and apply rule (3), renaming all occurrences of X_2 to X_3:

$$\underset{D}{[\text{NUM}: X_3]} \quad \underset{N}{[\text{NUM}: X_3]} \quad \underset{V}{[\text{NUM}: X_3]}$$

Example: (continued)
The leftmost element is selected, and (the terminal) rule (10) is applied, binding X_3 to pl:

$$\underset{two}{[num: X_3(pl)]} \quad \underset{N}{[\text{NUM}: X_3]} \quad \underset{V}{[\text{NUM}: X_3]}$$

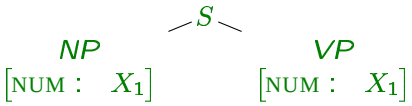
In the same way, rule (6) can be applied to the middle element in this form, and rule (8) to the rightmost, resulting in:

$$\underset{two}{[\text{NUM}: X_3(pl)]} \quad \underset{sheep}{[\text{NUM}: X_3(pl)]} \quad \underset{sleep}{[\text{NUM}: X_3(pl)]}$$

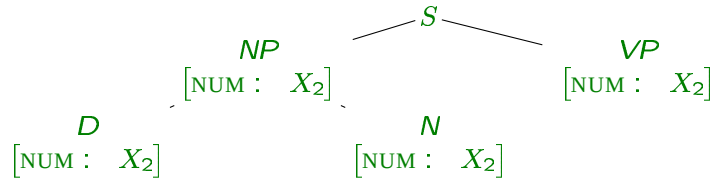
Thus the string two sheep sleep is indeed a sentence.

Derivation trees

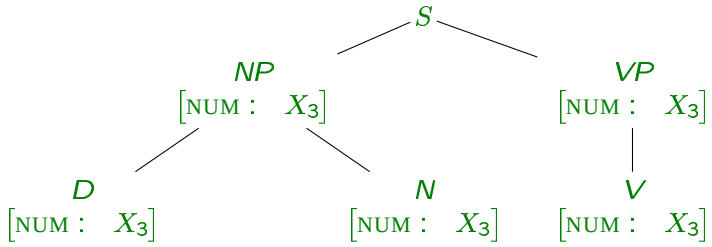
Example: Snapshots of a derivation sequence
We begin with the start symbol, the extended category S, which is expanded by applying rule (1), yielding (after renaming):



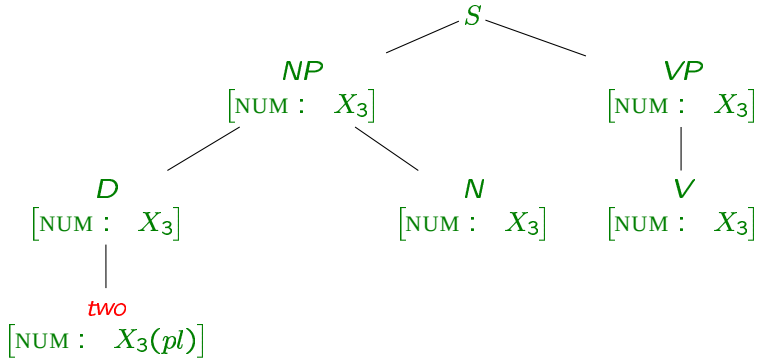
Example: (continued)
The next step is the application of rule (2) to the leftmost element in the frontier of the tree. Since this application results in binding X_1 with X_2, we rename all occurrences of X_1 in the tree to X_2, obtaining the following tree:



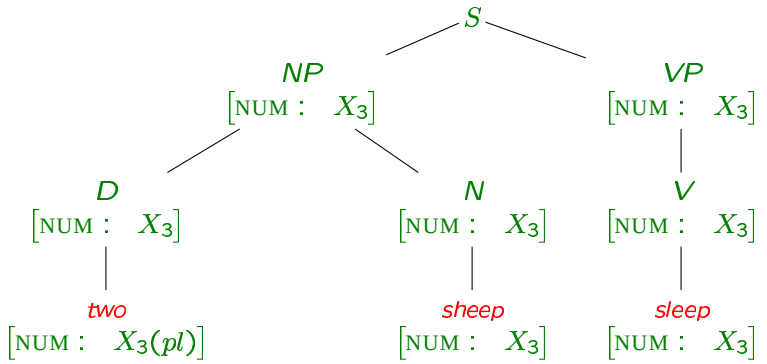
Example: (continued)
Now select the rightmost element in the frontier of the above tree and apply rule (3), renaming all occurrences of X_2 in the tree to X_3; the following tree is obtained:



Example: (continued)
Next, the leftmost element is selected, and (the terminal) rule (10) is applied, binding X_3 to pl:



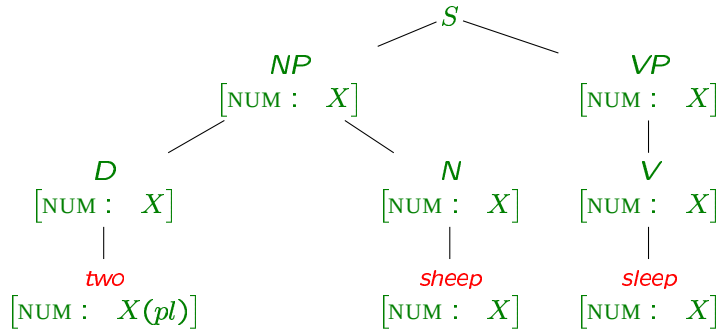
Example: (continued)
Similarly, rule (6) can be applied to the middle element in the frontier, and rule (8) to the rightmost, yielding:



Derivation trees

The final derivation tree for the same sentence:

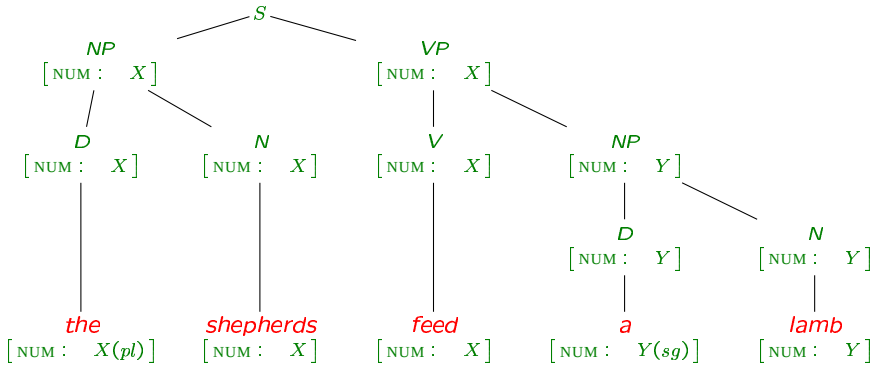
Example: Derivation tree



Derivation trees

The final derivation tree for the sentence the shepherds feed a lamb:

Example: Derivation tree



Languages

To determine whether a sequence of words, $w = a_1 \cdots a_n$, is in $L(G)$, consider a derivation in G whose first form consists of the start symbol (an extended category, viewed as an extended form of length 1), and whose last form is $\langle w, \sigma' \rangle$.

Let $\langle w, \sigma \rangle$ be an extended form obtained by concatenating $A_1, \ldots, A_n$, where each A_i is a lexical entry of the word a_1.

We say that $w \in L(G)$ if and only if σ' be a multi-AVM that is unifiable with σ: $\sigma \sqcup \sigma'$ does not fail.

Languages

Example: Language
Given this definition, observe, for example, that the string two sheep sleep is in the language generated by the example grammar G_1; we have seen a derivation sequence for this string. The first and the last elements of this sequence, namely the feature structures associated with the words two and sleep, are identical to lexical entries of G_1. However, the middle element, namely the feature structure associated with sheep, is more specific than (subsumed by) the lexical entry of sheep.

Languages

The language generated by the grammar G_1 is context free:

Example: A context-free grammar G'_1

$S \rightarrow S_{sg} \mid S_{pl}$

$S_{sg} \rightarrow NP_{sg}\ VP_{sg}$
$NP_{sg} \rightarrow D_{sg}\ N_{sg}$
$VP_{sg} \rightarrow V_{sg}$
$VP_{sg} \rightarrow V_{sg}\ NP_{sg} \mid V_{sg}\ NP_{pl}$
$D_{sg} \rightarrow a$
$N_{sg} \rightarrow lamb \mid sheep \mid \cdots$
$V_{sg} \rightarrow sleeps \mid \cdots$

$S_{pl} \rightarrow NP_{pl}\ VP_{pl}$
$NP_{pl} \rightarrow D_{pl}\ N_{pl}$
$VP_{pl} \rightarrow V_{pl}$
$VP_{pl} \rightarrow V_{pl}\ NP_{sg} \mid V_{pl}\ NP_{pl}$
$D_{pl} \rightarrow two$
$N_{pl} \rightarrow lambs \mid sheep \mid \cdots$
$V_{pl} \rightarrow sleep \mid \cdots$

Imposing case control

The extensions of the CFG formalism can be used for imposing various constraints on generated languages. Here we suggest a solution for the problem of controlling the case of a noun phrase.

First, add pronouns to the grammar:

(2.1) $\begin{matrix} NP \\ [\text{NUM}: X] \end{matrix} \rightarrow \begin{matrix} D \\ [\text{NUM}: X] \end{matrix} \quad \begin{matrix} N \\ [\text{NUM}: X] \end{matrix}$

(2.2) $\begin{matrix} NP \\ [\text{NUM}: X] \end{matrix} \rightarrow \begin{matrix} PropN \\ [\text{NUM}: X] \end{matrix}$

(2.3) $\begin{matrix} NP \\ [\text{NUM}: X] \end{matrix} \rightarrow \begin{matrix} Pron \\ [\text{NUM}: X] \end{matrix}$

Imposing case control

Additionally, the following terminal rules are needed:

$\begin{matrix} PropN \\ [\text{NUM}: sg] \end{matrix} \rightarrow$ Jacob | Rachel |...

$\begin{matrix} Pron \\ [\text{NUM}: sg] \end{matrix} \rightarrow$ she | her |...

Imposing case control

The additional rules allow sentences such as

She herds the sheep

Jacob loves her

but also non-sentences such as

*Her herds the sheep

*Jacob loves she

Imposing case control

We add a feature, CASE, to the feature structures associated with nominal categories: nouns, pronouns, proper names and noun phrases.

What should the values of the CASE feature be?

Imposing case control

(11) $\begin{bmatrix}\text{NUM}: & X\\ \text{CASE}: & Y\end{bmatrix}_{\textit{PropN}} \rightarrow \begin{bmatrix}\text{NUM}: & X(sg)\\ \text{CASE}: & Y\end{bmatrix}_{\textit{Rachel}}$

(12) $\begin{bmatrix}\text{NUM}: & X\\ \text{CASE}: & Y\end{bmatrix}_{\textit{PropN}} \rightarrow \begin{bmatrix}\text{NUM}: & X(sg)\\ \text{CASE}: & Y\end{bmatrix}_{\textit{Jacob}}$

(13) $\begin{bmatrix}\text{NUM}: & X\\ \text{CASE}: & Y\end{bmatrix}_{\textit{Pron}} \rightarrow \begin{bmatrix}\text{NUM}: & X(sg)\\ \text{CASE}: & Y(nom)\end{bmatrix}_{\textit{she}}$

(14) $\begin{bmatrix}\text{NUM}: & X\\ \text{CASE}: & Y\end{bmatrix}_{\textit{Pron}} \rightarrow \begin{bmatrix}\text{NUM}: & X(sg)\\ \text{CASE}: & Y(acc)\end{bmatrix}_{\textit{her}}$

Imposing case control

Percolating the value of the CASE feature from the lexical entries to the category NP:

(2.1) $\begin{bmatrix}\text{NUM}: & X\\ \text{CASE}: & Y\end{bmatrix}_{\textit{NP}} \rightarrow \begin{bmatrix}\text{NUM}: & X\end{bmatrix}_{\textit{D}} \quad \begin{bmatrix}\text{NUM}: & X\\ \text{CASE}: & Y\end{bmatrix}_{\textit{N}}$

(2.2) $\begin{bmatrix}\text{NUM}: & X\\ \text{CASE}: & Y\end{bmatrix}_{\textit{NP}} \rightarrow \begin{bmatrix}\text{NUM}: & X\\ \text{CASE}: & Y\end{bmatrix}_{\textit{PropN}}$

(2.3) $\begin{bmatrix}\text{NUM}: & X\\ \text{CASE}: & Y\end{bmatrix}_{\textit{NP}} \rightarrow \begin{bmatrix}\text{NUM}: & X\\ \text{CASE}: & Y\end{bmatrix}_{\textit{Pron}}$

Imposing case control

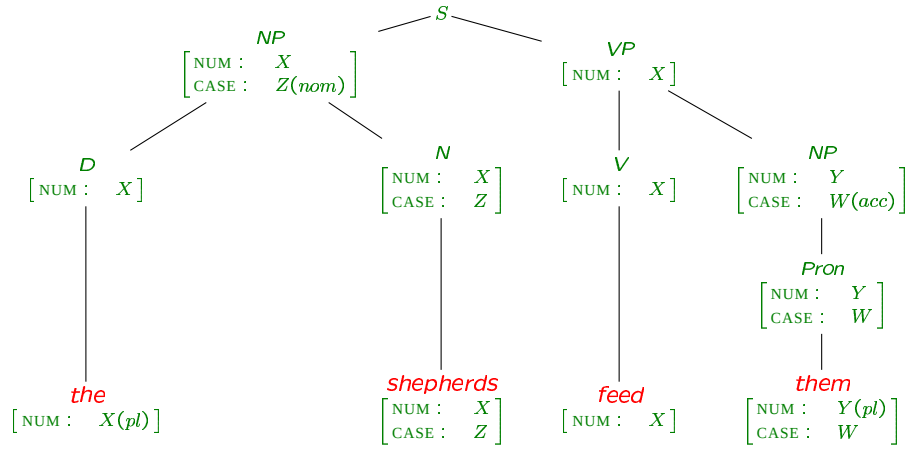
Imposing the constraint:

(1′) $S \rightarrow \underset{\left[\begin{smallmatrix}\text{NUM}: & X \\ \text{CASE}: & nom\end{smallmatrix}\right]}{NP} \quad \underset{\left[\text{NUM}: \; X\right]}{VP}$

(4′) $\underset{\left[\text{NUM}: \; X\right]}{VP} \rightarrow \underset{\left[\text{NUM}: \; X\right]}{V} \quad \underset{\left[\begin{smallmatrix}\text{NUM}: & Y \\ \text{CASE}: & acc\end{smallmatrix}\right]}{NP}$

Derivation tree with case control

Example: Derivation tree with case control



Example: (continued)
This tree represents a derivation which starts with the initial symbol, S, and ends with multi-AVM σ', where

$$\sigma' = \underset{\text{the}}{\left[\text{NUM}: \; X(pl)\right]} \; \underset{\text{shepherds}}{\left[\begin{smallmatrix}\text{NUM}: & X \\ \text{CASE}: & Z\end{smallmatrix}\right]} \; \underset{\text{feed}}{\left[\text{NUM}: \; X\right]} \; \underset{\text{them}}{\left[\begin{smallmatrix}\text{NUM}: & Y(pl) \\ \text{CASE}: & W(acc)\end{smallmatrix}\right]}$$

This multi-AVM is unifiable with (but not identical to!) the sequence of lexical entries of the words in the sentence, which is:

$$\sigma = \underset{\text{the}}{\left[\text{NUM}: \; [\,]\right]} \; \underset{\text{shepherds}}{\left[\begin{smallmatrix}\text{NUM}: & pl \\ \text{CASE}: & [\,]\end{smallmatrix}\right]} \; \underset{\text{feed}}{\left[\text{NUM}: \; pl\right]} \; \underset{\text{them}}{\left[\begin{smallmatrix}\text{NUM}: & pl \\ \text{CASE}: & acc\end{smallmatrix}\right]}$$

Imposing subcategorization constraints

We use the extended formalism for a naïve solution to the subcategorization problem; reminder:

intransitive verbs: sleep, walk, run, laugh, ...

transitive verbs (with a nominal object): feed, love, eat, ...

Imposing subcategorization constraints

First, the lexical entries of verbs are extended:

Example: Lexical entries for verbs

$$\overset{V}{\begin{bmatrix}\text{NUM}: & X\\ \text{SUBCAT}: & intrans\end{bmatrix}} \rightarrow \overset{sleeps}{\begin{bmatrix}\text{NUM}: & X(sg)\end{bmatrix}} \mid \overset{sleep}{\begin{bmatrix}\text{NUM}: & X(pl)\end{bmatrix}} \mid \ldots$$

$$\overset{V}{\begin{bmatrix}\text{NUM}: & X\\ \text{SUBCAT}: & trans\end{bmatrix}} \rightarrow \overset{feeds}{\begin{bmatrix}\text{NUM}: & X(sg)\end{bmatrix}} \mid \overset{feed}{\begin{bmatrix}\text{NUM}: & X(pl)\end{bmatrix}} \mid \ldots$$

Imposing subcategorization constraints

Second, the rules that involve verbs and verb phrases are extended:

Example: Modified rules for verb phrases

(4.1) $$\overset{VP}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}} \rightarrow \overset{V}{\begin{bmatrix}\text{NUM}: & X\\ \text{SUBCAT}: & intrans\end{bmatrix}}$$

(4.2) $$\overset{VP}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}} \rightarrow \overset{V}{\begin{bmatrix}\text{NUM}: & X\\ \text{SUBCAT}: & trans\end{bmatrix}} \quad \overset{NP}{\begin{bmatrix}\text{NUM}: & Y\end{bmatrix}}$$

Imposing subcategorization constraints

Example: Derivation of a shepherd feeds two sheep

$$S \overset{(1)}{\Rightarrow} \overset{NP}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}} \quad \overset{VP}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}}$$

$$\overset{(2)}{\Rightarrow} \overset{D}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}} \quad \overset{N}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}} \quad \overset{VP}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}}$$

$$\overset{(4.2)}{\Rightarrow} \overset{D}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}} \quad \overset{N}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}} \quad \overset{V}{\begin{bmatrix}\text{NUM}: & X\\ \text{SUBCAT}: & trans\end{bmatrix}} \quad \overset{NP}{\begin{bmatrix}\text{NUM}: & Y\end{bmatrix}}$$

Example: (continued)

$$\overset{(4.2)}{\Rightarrow} \overset{D}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}} \quad \overset{N}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}} \quad \overset{V}{\begin{bmatrix}\text{NUM}: & X\\ \text{SUBCAT}: & trans\end{bmatrix}} \quad \overset{NP}{\begin{bmatrix}\text{NUM}: & Y\end{bmatrix}}$$

$$\overset{(1)}{\Rightarrow} \overset{D}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}} \quad \overset{N}{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}} \quad \overset{V}{\begin{bmatrix}\text{NUM}: & X\\ \text{SUBCAT}: & trans\end{bmatrix}} \quad \overset{D}{\begin{bmatrix}\text{NUM}: & Y\end{bmatrix}} \quad \overset{}{[\text{NUM}}$$

$$\overset{*}{\Rightarrow} \overset{a}{\begin{bmatrix}\text{NUM}: & sg\end{bmatrix}} \quad \overset{shepherd}{\begin{bmatrix}\text{NUM}: & sg\end{bmatrix}} \quad \overset{feeds}{\begin{bmatrix}\text{NUM}: & sg\\ \text{SUBCAT}: & trans\end{bmatrix}} \quad \overset{two}{\begin{bmatrix}\text{NUM}: & pl\end{bmatrix}} \quad \overset{sh}{[\text{NUM}}$$

G_2, a complete E_0-grammar

Example: G_2, a complete E_0-grammar

$$S \rightarrow \begin{bmatrix}\text{NP}\\ \text{NUM}: X\\ \text{CASE}: nom\end{bmatrix} \begin{bmatrix}\text{VP}\\ \text{NUM}: X\end{bmatrix}$$

$$\begin{bmatrix}\text{NP}\\ \text{NUM}: X\\ \text{CASE}: Y\end{bmatrix} \rightarrow \begin{bmatrix}\text{D}\\ \text{NUM}: X\end{bmatrix} \begin{bmatrix}\text{N}\\ \text{NUM}: X\\ \text{CASE}: Y\end{bmatrix}$$

$$\begin{bmatrix}\text{NP}\\ \text{NUM}: X\\ \text{CASE}: Y\end{bmatrix} \rightarrow \begin{bmatrix}\text{Pron}\\ \text{NUM}: X\\ \text{CASE}: Y\end{bmatrix} \mid \begin{bmatrix}\text{PropN}\\ \text{NUM}: X\\ \text{CASE}: Y\end{bmatrix}$$

$$\begin{bmatrix}\text{VP}\\ \text{NUM}: X\end{bmatrix} \rightarrow \begin{bmatrix}\text{V}\\ \text{NUM}: X\\ \text{SUBCAT}: intrans\end{bmatrix}$$

$$\begin{bmatrix}\text{VP}\\ num: X\end{bmatrix} \rightarrow \begin{bmatrix}\text{V}\\ \text{NUM}: X\\ \text{SUBCAT}: trans\end{bmatrix} \begin{bmatrix}\text{NP}\\ \text{NUM}: Y\\ \text{CASE}: acc\end{bmatrix}$$

Example: (continued)

$$\begin{bmatrix}\text{N}\\ \text{NUM}: X\\ \text{CASE}: Y\end{bmatrix} \rightarrow \begin{bmatrix}\text{lamb}\\ \text{NUM}: X(sg)\\ \text{CASE}: Y\end{bmatrix} \mid \begin{bmatrix}\text{lambs}\\ \text{NUM}: X(pl)\\ \text{CASE}: Y\end{bmatrix} \mid \ldots$$

$$\begin{bmatrix}\text{Pron}\\ \text{NUM}: X\\ \text{CASE}: Y\end{bmatrix} \rightarrow \begin{bmatrix}\text{she}\\ \text{NUM}: X(sg)\\ \text{CASE}: Y(nom)\end{bmatrix} \mid \begin{bmatrix}\text{her}\\ \text{NUM}: X(sg)\\ \text{CASE}: Y(acc)\end{bmatrix} \mid \ldots$$

$$\begin{bmatrix}\text{PropN}\\ \text{NUM}: X\\ \text{CASE}: Y\end{bmatrix} \rightarrow \begin{bmatrix}\text{Rachel}\\ \text{NUM}: X(sg)\\ \text{CASE}: Y\end{bmatrix} \mid \begin{bmatrix}\text{Jacob}\\ \text{NUM}: X(sg)\\ \text{CASE}: Y\end{bmatrix} \mid \ldots$$

$$\begin{bmatrix}\text{V}\\ \text{NUM}: X\\ \text{SUBCAT}: intrans\end{bmatrix} \rightarrow \begin{bmatrix}\text{sleeps}\\ \text{NUM}: X(sg)\end{bmatrix} \mid \begin{bmatrix}\text{sleep}\\ \text{NUM}: X(pl)\end{bmatrix} \mid \ldots$$

$$\begin{bmatrix}\text{V}\\ \text{NUM}: X\\ \text{SUBCAT}: trans\end{bmatrix} \rightarrow \begin{bmatrix}\text{feeds}\\ \text{NUM}: X(sg)\end{bmatrix} \mid \begin{bmatrix}\text{feed}\\ \text{NUM}: X(pl)\end{bmatrix} \mid \ldots$$

$$\begin{bmatrix}\text{D}\\ \text{NUM}: X\end{bmatrix} \rightarrow \begin{bmatrix}\text{a}\\ \text{NUM}: X(sg)\end{bmatrix} \mid \begin{bmatrix}\text{two}\\ \text{NUM}: X(pl)\end{bmatrix} \mid \ldots$$

Internalizing categories

The grammars we have seen so far had an explicit context-free backbone (or skeleton), obtained by considering the (context-free) grammar induced by the base categories.

This is not imposed by the formalism; rather, the base categories might be internalized into the feature structures themselves.

Internalizing categories

For example, the rule

$$\begin{bmatrix}\text{NP}\\ \text{NUM}: X\end{bmatrix} \rightarrow \begin{bmatrix}\text{D}\\ \text{NUM}: X\end{bmatrix} \begin{bmatrix}\text{N}\\ \text{NUM}: X\end{bmatrix}$$

can be re-written as

$$\begin{bmatrix}\text{CAT}: np\\ \text{NUM}: X\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: d\\ \text{NUM}: X\end{bmatrix} \begin{bmatrix}\text{CAT}: n\\ \text{NUM}: X\end{bmatrix}$$

Internalizing categories

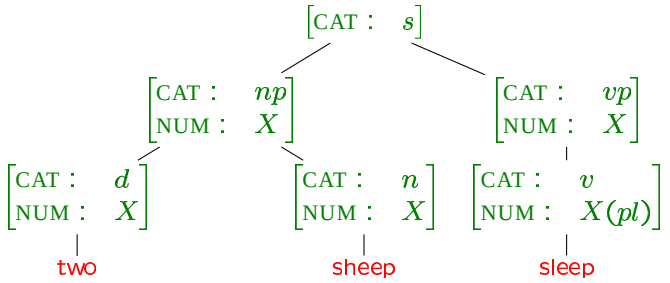
In the new presentation of grammars, productions are essentially multi-AVMs.

Derivations, derivation trees, languages...

Special features and the signature.

Internalizing categories

Example: Derivation tree



Internalizing categories

Once the base category of a phrase is admitted as the value of one of the features in the feature structure associated with that phrase, it does not have to be represented as an atomic value.

Internalizing categories

For example, the Chomskian representation of categories:

nouns: $\begin{bmatrix} \text{N} : & + \\ \text{V} : & - \end{bmatrix}$

verbs: $\begin{bmatrix} \text{N} : & - \\ \text{V} : & + \end{bmatrix}$

adjectives: $\begin{bmatrix} \text{N} : & + \\ \text{V} : & + \end{bmatrix}$

prepositions: $\begin{bmatrix} \text{N} : & - \\ \text{V} : & - \end{bmatrix}$

Internalizing categories

Internalization of the category results in additional expressive power.

It now becomes possible to consider feature structures in which the value of the CAT feature is underspecified, or even unrestricted.

For example, one might describe a phrase in singular using the feature structure

$$\begin{bmatrix} \text{CAT}: & [\,] \\ \text{NUM}: & sg \end{bmatrix}$$

Internalizing categories

Once information about the category of a phrase is embedded within the feature structure, it can be manipulated in more ways than simply encoding the category of a phrase.

Internalized categories will be used to:

- represent information about the subcategories of verbs
- list information about constituents that are "moved", or "transformed", using the *slash* notation
- account for coordination.

Subcategorization lists

Motivation: to account for the subcategorization data in a more general, elegant way, extending the coverage of our grammar from the smallest fragment E_0 to the fragment E_1.

In E_1 different verbs subcategorize for different kinds of complements: noun phrases, infinitival verb phrases, sentences etc. Also, some verbs require more than one complement.

The idea behind the solution is to store in the lexical entry of each verb not an atomic feature indicating its subcategory, but rather a list of atomic categories, indicating the appropriate complements of the verb.

Subcategorization lists

Example: Lexical entries of verbs using subcategorization lists

sleep $\begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & elist \\ \text{NUM}: & pl \end{bmatrix}$

love $\begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \langle [\,\text{CAT}: \quad np\,] \rangle \\ \text{NUM}: & pl \end{bmatrix}$

give $\begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \langle [\,\text{CAT}: \quad np\,], [\,\text{CAT}: \quad np\,] \rangle \\ \text{NUM}: & pl \end{bmatrix}$

tell $\begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \langle [\,\text{CAT}: \quad np\,], [\,\text{CAT}: \quad s\,] \rangle \\ \text{NUM}: & pl \end{bmatrix}$

Subcategorization lists

The grammar rules must be modified to reflect the additional wealth of information in the lexical entries:

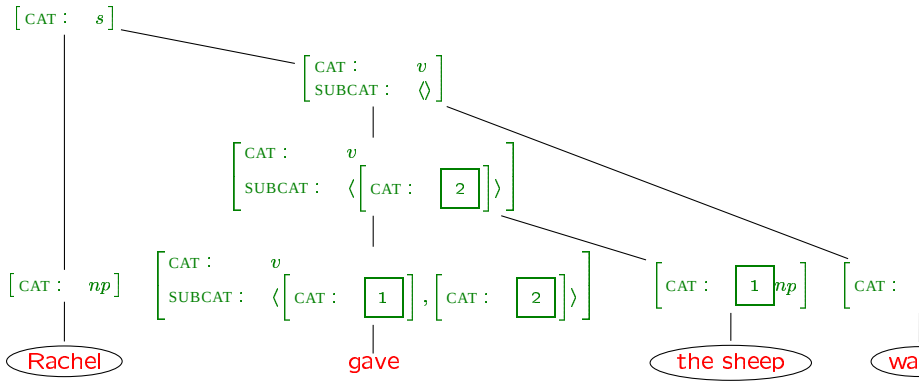
Example: VP rules using subcategorization lists

$$\begin{bmatrix}\text{CAT}: & s\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & np\end{bmatrix} \begin{bmatrix}\text{CAT}: & v \\ \text{SUBCAT}: & elist\end{bmatrix}$$

$$\begin{bmatrix}\text{CAT}: & v \\ \text{SUBCAT}: & Y\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & v \\ \text{SUBCAT}: & \begin{bmatrix}\text{FIRST}: & \begin{bmatrix}\text{CAT}: & X\end{bmatrix} \\ \text{REST}: & Y\end{bmatrix}\end{bmatrix} \begin{bmatrix}\text{CAT}: & X\end{bmatrix}$$

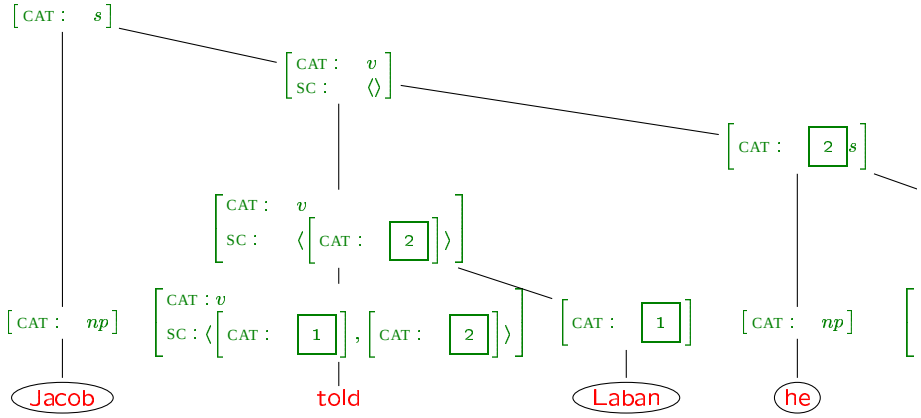
Subcategorization lists

Example: A derivation tree



Subcategorization lists

Example: A derivation tree



Subcategorization lists

In the above grammar, categories on subcategorization lists are represented as an atomic symbol.

The method outlined here can be used with more complex encodings of categories. In other words, the specification of categories in a subcategorization list can include all the constraints that the verb imposes on its complements

Subcategorization lists

Example: Subcategorization imposes case constraints

	Ich	gebe	dem	Hund	den	Knochen
	I	*give*	*the(dat)*	*dog*	*the(acc)*	*bone*
	I give the dog the bone					
*	Ich	gebe	den	Hund	den	Knochen
	I	*give*	*the(acc)*	*dog*	*the(acc)*	*bone*
*	Ich	gebe	dem	Hund	dem	Knochen
	I	*give*	*the(dat)*	*dog*	*the(dat)*	*bone*

Subcategorization lists

The lexical entry of gebe, then, could be:

$$\begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \left\langle \begin{bmatrix} \text{CAT}: & np \\ \text{CASE}: & dat \end{bmatrix}, \begin{bmatrix} \text{CAT}: & np \\ \text{CASE}: & acc \end{bmatrix} \right\rangle \\ \text{NUM}: & sg \end{bmatrix}$$

The VP rule has to be slightly modified:

$$\begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & Y \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \begin{bmatrix} \text{FIRST}: & X \\ \text{REST}: & Y \end{bmatrix} \end{bmatrix} X([\,])$$

G_3, a complete E_1-grammar

Example: G_3, a complete E_1-grammar

$$[\text{CAT}: s] \rightarrow \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & X \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & X \\ \text{SUBCAT}: & elist \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & X \\ \text{SUBCAT}: & Y \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & X \\ \text{SUBCAT}: & \begin{bmatrix} \text{FIRST}: & Z \\ \text{REST}: & Y \end{bmatrix} \end{bmatrix} Z([\,])$$

$$\begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & X \\ \text{CASE}: & Y \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & X \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & X \\ \text{CASE}: & Y \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & X \\ \text{CASE}: & Y \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & X \\ \text{CASE}: & Y \end{bmatrix} \mid \begin{bmatrix} \text{CAT}: & propn \\ \text{NUM}: & X \\ \text{CASE}: & Y \end{bmatrix}$$

Example: (continued)

sleep $\rightarrow$ $\begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & elist \\ \text{NUM}: & pl \end{bmatrix}$

love $\rightarrow$ $\begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \left\langle \begin{bmatrix} \text{CAT}: & np \\ \text{CASE}: & acc \end{bmatrix} \right\rangle \\ \text{NUM}: & pl \end{bmatrix}$

lamb $\rightarrow$ $\begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & sg \\ \text{CASE}: & Y \end{bmatrix}$

give $\rightarrow$ $\begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \langle \begin{bmatrix} \text{CAT}: & np \\ \text{CASE}: & acc \end{bmatrix}, [\text{CAT}: \\ \text{NUM}: & pl \end{bmatrix}$

tell $\rightarrow$ $\begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \langle \begin{bmatrix} \text{CAT}: & np \\ \text{CASE}: & acc \end{bmatrix}, [\text{CAT}: \\ \text{NUM}: & pl \end{bmatrix}$

lambs $\rightarrow$ $\begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & Y \end{bmatrix}$

Example: (continued)

$$\text{she} \rightarrow \begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & sg \\ \text{CASE}: & nom \end{bmatrix} \quad \text{her} \rightarrow \begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & pl \\ \text{CASE}: & acc \end{bmatrix}$$

$$\text{Rachel} \rightarrow \begin{bmatrix} \text{CAT}: & propn \\ \text{NUM}: & sg \end{bmatrix} \quad \text{Jacob} \rightarrow \begin{bmatrix} \text{CAT}: & propn \\ \text{NUM}: & sg \end{bmatrix}$$

$$\text{a} \rightarrow \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & sg \end{bmatrix} \quad \text{two} \rightarrow \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & pl \end{bmatrix}$$

Long distance dependencies

Internalized categories are very useful in the treatment of unbounded dependencies, which are included in the grammar fragment E_3.

Such phenomena involve a "missing" constituent that is realized outside the clause from which it is missing, as in:

(1) The shepherd wondered whom Jacob loved ⌙.

(2) The shepherd wondered whom Laban thought Jacob loved ⌙.

(3) The shepherd wondered whom Laban thought Rachel claimed Jacob loved ⌙.

Long distance dependencies

An attempt to replace the gap with an explicit noun phrase results in ungrammaticality:

(4) *The shepherd wondered who Jacob loved Rachel.

Long distance dependencies

The gap need not be in the object position:

(5) Jacob wondered who ⌙ loved Leah

(6) Jacob wondered who Laban believed ⌙ loved Leah

Again, an explicit noun phrase filling the gap results in ungrammaticality:

(7) Jacob wondered who the shepherd loved Leah

Long distance dependencies

More than one gap may be present in a sentence (and, hence, more than one filler):

(8a) This is the well which Jacob is likely to ␣ draw water from ␣

(8b) It was Leah that Jacob worked for ␣ without loving ␣

In some languages (e.g., Norwegian) there is no (principled) bound on the number of gaps that can occur in a single clause.

Long distance dependencies

There are other fragments of English in which long distance dependencies are manifested in other forms. *Topicalization*:

(9) Rachel, Jacob loved ␣

(10) Rachel, every shepherd knew Jacob loved ␣

Another example is *interrogative sentences*:

(11) Who did Jacob love ␣?

(12) Who did Laban believe Jacob loved ␣?

We do not account for such phenomena here.

Long distance dependencies

Phrases such as whom Jacob loved ␣ or who ␣ loved Rachel are instances of a category that we haven't discussed yet.

They are basically *sentences*, with a constituent which is "moved" from its default position and realized as a wh-pronoun in front of the phrase.

We will represent such phrases by using the same category, *s*, which we used for sentences; but to distinguish them from declarative sentences we will add a feature, QUE, to the category. The value of QUE will be '+' in sentences with an interrogative pronoun realizing a transposed constituent.

Long distance dependencies

We add a lexical entry for the pronoun whom:

$$\text{whom} \rightarrow \begin{bmatrix} \text{CAT}: & pron \\ \text{CASE}: & acc \\ \text{QUE}: & + \end{bmatrix}$$

and update the rule that derives pronouns:

$$\begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & X \\ \text{CASE}: & Y \\ \text{QUE}: & Q \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & X \\ \text{CASE}: & Y \\ \text{QUE}: & Q \end{bmatrix}$$

Long distance dependencies

We now propose an extension of G_3 that can handle long distance dependencies.

The idea is to allow partial phrases, such as Jacob loved ␣, to be derived from a category that is similar to the category of the full phrase, in this case Jacob loved Rachel; but to signal in some way that a constituent, in this case a noun phrase, is missing.

We extend G_3 with two additional rules, based on the first two rules of G_3.

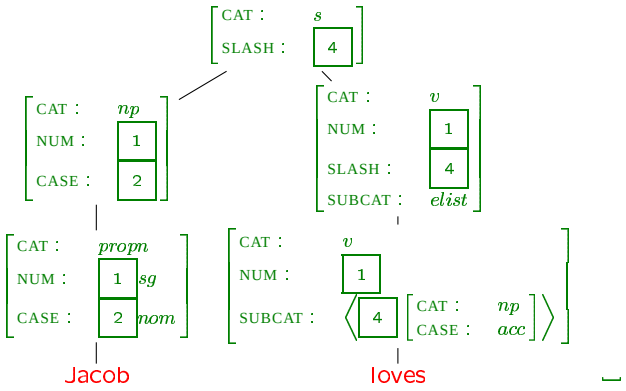
Long distance dependencies

(3) $$\begin{bmatrix}\text{CAT}: & s\\ \text{SLASH}: & Z\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & np\\ \text{NUM}: & X\\ \text{CASE}: & nom\end{bmatrix} \begin{bmatrix}\text{CAT}: & v\\ \text{NUM}: & X\\ \text{SUBCAT}: & elist\\ \text{SLASH}: & Z\end{bmatrix}$$

(4) $$\begin{bmatrix}\text{CAT}: & v\\ \text{NUM}: & X\\ \text{SUBCAT}: & Y\\ \text{SLASH}: & Z\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & v\\ \text{NUM}: & X\\ \text{SUBCAT}: & \begin{bmatrix}\text{FIRST}: & Z\\ \text{REST}: & Y\end{bmatrix}\end{bmatrix}$$

Long distance dependencies

Example: A derivation tree for Jacob loves ␣



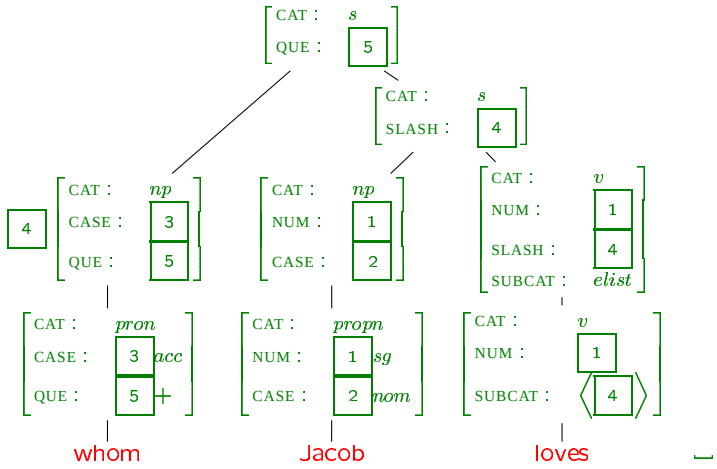
Long distance dependencies

Now that partial phrases can be derived, with a record of their "missing" constituent, all that is needed is a rule for creating "complete" sentences by combining the missing category with a "slashed" sentence:

(5) $$\begin{bmatrix}\text{CAT}: & s\\ \text{QUE}: & Q\end{bmatrix} \rightarrow Z(\begin{bmatrix}\text{QUE}: & Q(+)\end{bmatrix}) \begin{bmatrix}\text{CAT}: & s\\ \text{SLASH}: & Z\end{bmatrix}$$

Long distance dependencies

Example: A derivation tree for whom Jacob loves ␣



Long distance dependencies

Unbounded dependencies can hold across several clause boundaries:

The shepherd wondered whom Jacob loved ␣.

The shepherd wondered whom Laban thought Jacob loved ␣.

The shepherd wondered whom Laban thought Leah claimed Jacob loved ␣.

Also, the dislocated constituent does not have to be an object:

The shepherd wondered who ␣ loved Rachel.

The shepherd wondered who Laban thought ␣ loved Rachel.

The shepherd wondered who Laban thought Leah claimed ␣ loved Rachel.

Long distance dependencies

The solution we proposed for the simple case of unbounded dependencies can be easily extended to the more complex examples:

- a slash introduction rule;
- slash propagation rules;
- and a gap filler rule.

In order to account for filler-gap relations that hold across several clauses, all that needs to be done is to add more slash propagation rules.

Long distance dependencies

For example, in

The shepherd wondered whom Laban thought Jacob loved ␣.

the slash is introduced by the verb phrase loved ␣, and is propagated to the sentence Jacob loved ␣ by rule (3).

A rule that propagates the value of SLASH from a sentential object to the verb phrase of which it is an object:

$$(6)\quad \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & X \\ \text{SUBCAT}: & Y \\ \text{SLASH}: & Z \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & X \\ \text{SUBCAT}: & \begin{bmatrix} \text{FIRST}: & W \\ \text{REST}: & Y \end{bmatrix} \end{bmatrix} W(\begin{bmatrix} \text{SLASH}: & Z \end{bmatrix})$$

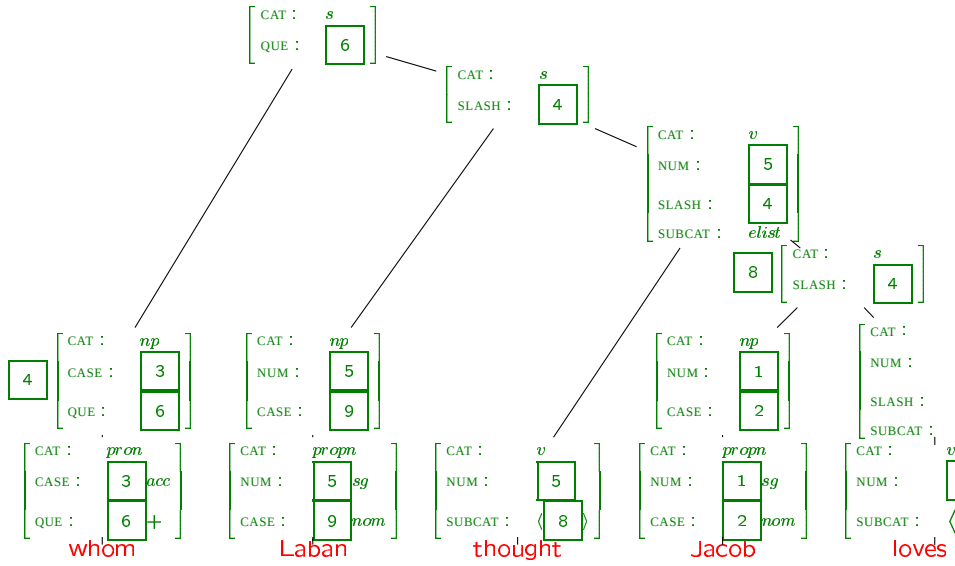
Long distance dependencies

Then, the slash is propagated from the verb phrase thought Jacob loved ␣ to the sentence Laban thought Jacob loved ␣:

$$(7)\quad \begin{bmatrix} \text{CAT}: & s \\ \text{SLASH}: & Z \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & X \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & X \\ \text{SUBCAT}: & elist \\ \text{SLASH}: & Z \end{bmatrix}$$

Long distance dependencies

Example: A derivation tree for whom Laban thought Jacob loves ␣



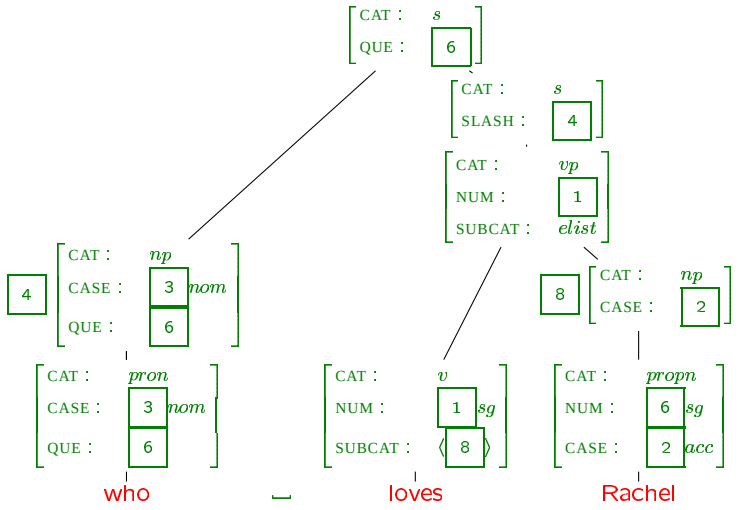
Long distance dependencies

Finally, to account for gaps in the subject position, all that is needed is an additional slash introduction rule:

(8)
$$\begin{bmatrix} \text{CAT}: & s \\ \text{SLASH}: & \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & X \\ \text{CASE}: & nom \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & X \\ \text{SUBCAT}: & elist \end{bmatrix}$$

Long distance dependencies

Example: A derivation tree for who ␣ loves Rachel



Subject and object control

Subject and object control phenomena capture the differences between the 'understood' subjects of the infinitive verb phrase to work seven years in the following sentences:

Jacob promised Laban to work seven years

Laban persuaded Jacob to work seven years

Subject and object control

The key observation in the solution is that the differences between the two examples stem from differences in the matrix verbs:

- promise is a *subject control* verb
- persuade is *object control*.

Subject and object control

Our departure point is the grammar G_3. We modify it by adding a SUBJ feature to verb phrases, whose value is a feature structure associated with the phrase that serves as the verb's subject.

Subject and object control

Example: G_4: explicit SUBJ values

$$\begin{bmatrix}\text{CAT}: & s\end{bmatrix} \rightarrow \boxed{1}\begin{bmatrix}\text{CAT}: & np\\ \text{CASE}: & nom\\ \text{NUM}: & \boxed{7}\end{bmatrix}\begin{bmatrix}\text{CAT}: & v\\ \text{NUM}: & \boxed{7}\\ \text{SUBCAT}: & elist\\ \text{SUBJ}: & \boxed{1}\end{bmatrix}$$

$$\begin{bmatrix}\text{CAT}: & v\\ \text{NUM}: & \boxed{7}\\ \text{SUBCAT}: & \boxed{4}\\ \text{SUBJ}: & \boxed{1}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & v\\ \text{NUM}: & \boxed{7}\\ \text{SUBCAT}: & \begin{bmatrix}\text{FIRST}: & \boxed{2}\\ \text{REST}: & \boxed{4}\end{bmatrix}\\ \text{SUBJ}: & \boxed{1}\end{bmatrix}\boxed{2}[\,]$$

Example: (continued)

$$\begin{bmatrix}\text{CAT}: & np\\ \text{NUM}: & \boxed{7}\\ \text{CASE}: & \boxed{6}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & d\\ \text{NUM}: & \boxed{7}\end{bmatrix} \begin{bmatrix}\text{CAT}: & n\\ \text{NUM}: & \boxed{7}\\ \text{CASE}: & \boxed{6}\end{bmatrix}$$

$$\begin{bmatrix}\text{CAT}: & np\\ \text{NUM}: & \boxed{7}\\ \text{CASE}: & \boxed{6}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & pron\\ \text{NUM}: & \boxed{7}\\ \text{CASE}: & \boxed{6}\end{bmatrix} \mid \begin{bmatrix}\text{CAT}: & propn\\ \text{NUM}: & \boxed{7}\\ \text{CASE}: & \boxed{6}\end{bmatrix}$$

Example: (continued)

sleep →
$$\begin{bmatrix}\text{CAT}: & v\\ \text{SUBCAT}: & elist\\ \text{SUBJ}: & \begin{bmatrix}\text{CAT}: & np\\ \text{CASE}: & nom\end{bmatrix}\\ \text{NUM}: & pl\end{bmatrix}$$

love →
$$\begin{bmatrix}\text{CAT}: & v\\ \text{SUBCAT}: & \langle\begin{bmatrix}\text{CAT}: & np\\ \text{CASE}: & acc\end{bmatrix}\rangle\\ \text{SUBJ}: & \begin{bmatrix}\text{CAT}: & np\\ \text{CASE}: & nom\end{bmatrix}\\ \text{NUM}: & pl\end{bmatrix}$$

give →
$$\begin{bmatrix}\text{CAT}: & v\\ \text{SUBCAT}: & \langle\begin{bmatrix}\text{CAT}: & np\\ \text{CASE}: & acc\end{bmatrix}, \begin{bmatrix}\text{CAT}: & np\end{bmatrix}\rangle\\ \text{SUBJ}: & \begin{bmatrix}\text{CAT}: & np\\ \text{CASE}: & nom\end{bmatrix}\\ \text{NUM}: & pl\end{bmatrix}$$

Example: (continued)

lamb → $\begin{bmatrix}\text{CAT}: & n\\ \text{NUM}: & sg\\ \text{CASE}: & \boxed{6}\end{bmatrix}$ lambs → $\begin{bmatrix}\text{CAT}: & n\\ \text{NUM}: & pl\\ \text{CASE}: & \boxed{6}\end{bmatrix}$

she → $\begin{bmatrix}\text{CAT}: & pron\\ \text{NUM}: & sg\\ \text{CASE}: & nom\end{bmatrix}$ her → $\begin{bmatrix}\text{CAT}: & pron\\ \text{NUM}: & pl\\ \text{CASE}: & acc\end{bmatrix}$

Rachel → $\begin{bmatrix}\text{CAT}: & propn\\ \text{NUM}: & sg\end{bmatrix}$ Jacob → $\begin{bmatrix}\text{CAT}: & propn\\ \text{NUM}: & sg\end{bmatrix}$

a → $\begin{bmatrix}\text{CAT}: & d\\ \text{NUM}: & sg\end{bmatrix}$ two → $\begin{bmatrix}\text{CAT}: & d\\ \text{NUM}: & pl\end{bmatrix}$

Subject and object control

Accounting for infinitival verb phrases:

to work →
$$\begin{bmatrix}\text{CAT}: & v\\ \text{VFORM}: & inf\\ \text{SUBCAT}: & elist\\ \text{SUBJ}: & \begin{bmatrix}\text{CAT}: & np\end{bmatrix}\end{bmatrix}$$

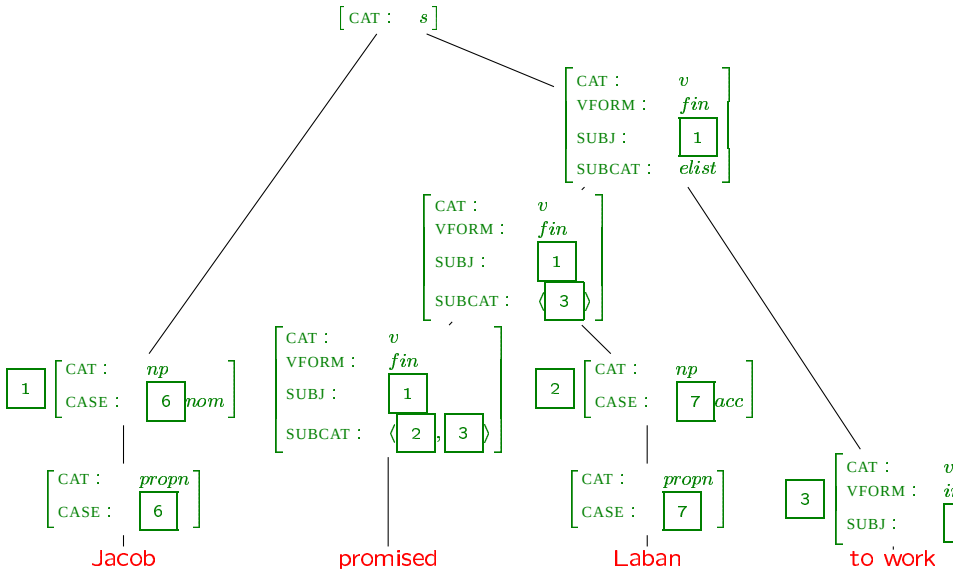
Subject and object control

The lexical entries of verbs such as promise or persuade:

$$\text{promised} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{VFORM}: & fin \\ \text{SUBCAT}: & \langle \begin{bmatrix} \text{CAT}: & np \\ \text{CASE}: & acc \end{bmatrix}, \begin{bmatrix} \text{CAT}: & v \\ \text{VFORM}: & inf \\ \text{SUBJ}: & \boxed{1} \end{bmatrix} \rangle \\ \text{SUBJ}: & \boxed{1} \begin{bmatrix} \text{CAT}: & np \\ \text{CASE}: & nom \end{bmatrix} \\ \text{NUM}: & [\,] \end{bmatrix}$$

Subject and object control

Example: A derivation tree for Jacob promised Laban to work



Subject and object control

The only difference between the lexical entries of promised and persuaded is that in the latter, the value of the SUBJ list of the infinitival verb phrase is reentrant with the first element on the SUBCAT list of the matrix verb, rather than with its SUBJ value:

$$\text{persuaded} \rightarrow \begin{bmatrix} \text{CAT} : & v \\ \text{VFORM} : & fin \\ \text{SUBCAT} : & \langle \boxed{1} \begin{bmatrix} \text{CAT} : & np \\ \text{CASE} : & acc \end{bmatrix}, \begin{bmatrix} \text{CAT} : & v \\ \text{VFORM} : & inf \\ \text{SUBJ} : & \boxed{1} \end{bmatrix} \rangle \\ \text{SUBJ} : & \begin{bmatrix} \text{CAT} : & np \\ \text{CASE} : & nom \end{bmatrix} \\ \text{NUM} : & [\,] \end{bmatrix}$$

Constituent coordination

Many languages exhibit a phenomenon by which constituents of the same category can be conjoined to form a constituent of this category.

Constituent coordination

N: no man lift up his [hand] or [foot] in all the land of Egypt

NP: Jacob saw [Rachel] and [the sheep of Laban]

VP: Jacob [went on his journey] and [came to the land of the people of the east]

VP: Jacob [went near], and [rolled the stone from the well's mouth], and [watered

ADJ: every [speckled] and [spotted] sheep

ADJP: Leah was [tender eyed] but [not beautiful]

S: [Leah had four sons], but [Rachel was barren]

S: she said to Jacob, "[Give me children], or [I shall die]!"

Constituent coordination

We extend the grammar fragment to cover coordination, referring to it as E_4.

The lexicon of a grammar for E_4 is extended by a closed class of conjunction words; categorized under Conj, this class includes the words and, or, but and perhaps a few others (E_4 contains only these three).

We assume that in E_4, every category of E_0 can be conjoined.

We also assume – simplifying a little – that the same conjunctions are possible for all the categories.

Constituent coordination

A context-free grammar for coordination:

$$\begin{array}{lcl} S & \rightarrow & S\ Conj\ S \\ NP & \rightarrow & NP\ Conj\ NP \\ VP & \rightarrow & VP\ Conj\ VP \\ \vdots & & \\ Conj & \rightarrow & \text{and, or, but, } \ldots \end{array}$$

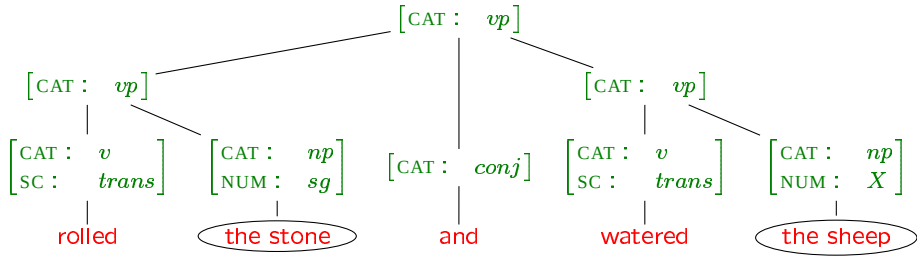
Constituent coordination

With generalized categories, a single production is sufficient:

$$\left[\text{CAT}: \quad X\right] \rightarrow \left[\text{CAT}: \quad X\right] \left[\text{CAT}: \quad conj\right] \left[\text{CAT}: \quad X\right]$$

Constituent coordination

Example: Coordination



Constituent coordination

The above solution is over-simplifying:

- it allows coordination of E_0 categories, but also of E_4 categories;
- it does not handle the properties of coordinated phrases properly;
- it does not permit conjunction of unlikes and of non-constituents.

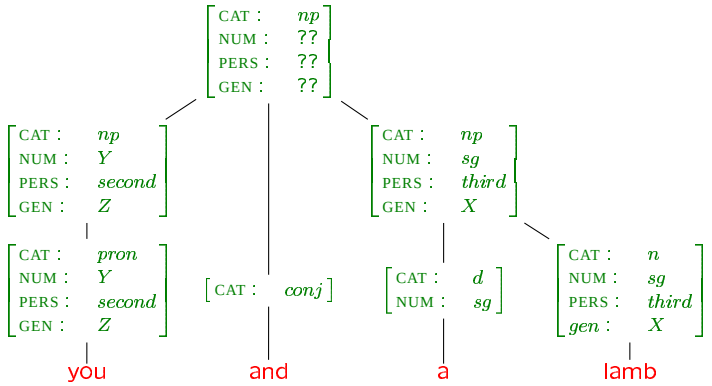
Constituent coordination

Not every category can be coordinated: for example, in English conjunctions cannot themselves be conjoined (in most cases):

$$\begin{bmatrix}\text{CAT}: & X\\ \text{CONJOINABLE}: & -\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & X\\ \text{CONJOINABLE}: & +\end{bmatrix} \begin{bmatrix}\text{CAT}: & conj\end{bmatrix} \begin{bmatrix}\text{CAT}: & X\\ \text{CONJOINABLE}: & +\end{bmatrix}$$

Properties of conjoined constituents

Example: NP coordination



Coordination of unlikes

Consider the following English data:

Joseph became wealthy

Joseph became a minister

Joseph became [wealthy and a minister]

Joseph grew wealthy

*Joseph grew a minister

*Joseph grew [wealthy and a minister]

Coordination of unlikes

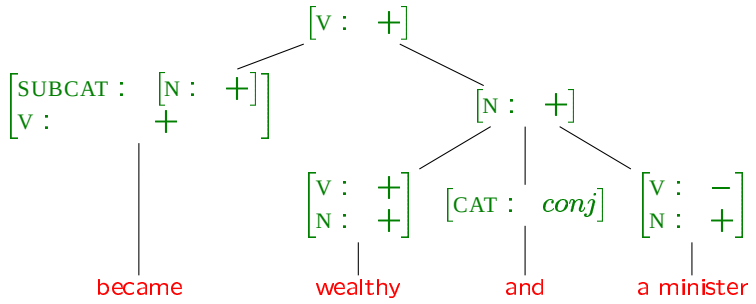
These data are easy to account for in a unification-based framework with a possibility of specifying generalization instead of unification in certain cases:

$$\boxed{1} \sqcap \boxed{2} \rightarrow \boxed{1}\begin{bmatrix}\text{CAT}: & X\end{bmatrix} \begin{bmatrix}\text{CAT}: & conj\end{bmatrix} \boxed{2}\begin{bmatrix}\text{CAT}: & Y\end{bmatrix}$$

where '$\sqcap$' is the generalization operator.

Coordination of unlikes

Example:



Coordination of unlikes

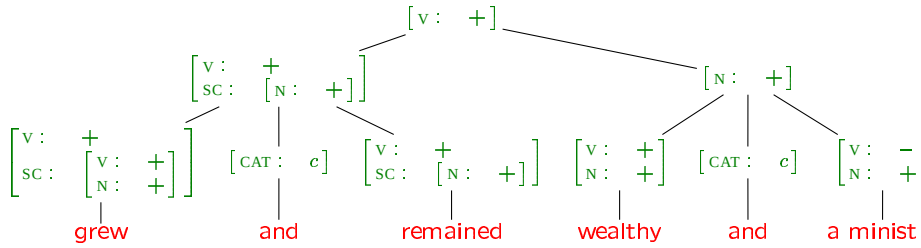
The situation becomes more complicated when verbs, too, are conjoined:

*Joseph [grew and remained] [wealthy and a minister]

While this example is ungrammatical, it is obtainable by the same grammar

Coordination of unlikes

Example:



Non-constituent coordination

Rachel gave the sheep [grass] and [water]

Rachel gave [the sheep grass] and [the lambs water]

Rachel [kissed] and Jacob [hugged] Binyamin

Expressiveness of unification grammars

Just how expressive are unification grammars?

What is the class of languages generated by unification grammars?

Trans-context-free languages

A grammar, G_{abc}, for the language $L = \{a^n b^n c^n \mid n > 0\}$.

Feature structures will have two features: CAT, which stands for category, and T, which "counts" the length of sequences of a-s, b-s and c-s.

The "category" is ap for strings of a-s, bp for b-s and cp for c-s. The categories *at*, *bt* and *ct* are pre-terminal categories of the words a, b and c, respectively.

"Counting" is done in unary base: a string of length n is derived by an AVM (that is, an multi-AVM of length 1) whose depth is n.

For example, the string bbb is derived by the following AVM:

$$\begin{bmatrix} \text{CAT}: & bp \\ \text{T}: & \begin{bmatrix} \text{T}: & \begin{bmatrix} \text{T}: & end \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Trans-context-free languages

Example: A unification grammar G_{abc} for the language $\{a^n b^n c^n \mid n > 0\}$

The signature of the grammar consists in the features CAT and T and the atoms *s*, *ap*, *bp*, *cp*, *at*, *bt*, *ct* and *end*. The terminal symbols are, of course, *a*, *b* and *c*. The start symbol is the left-hand side of the first rule.

$$\rho_1: \quad \begin{bmatrix} \text{CAT}: & s \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \text{CAT}: & ap \\ \text{T}: & X \end{bmatrix} \quad \begin{bmatrix} \text{CAT}: & bp \\ \text{T}: & X \end{bmatrix} \quad \begin{bmatrix} \text{CAT}: & cp \\ \text{T}: & X \end{bmatrix}$$

Example: (continued)

$$\rho_2: \quad \begin{bmatrix} \text{CAT}: & ap \\ \text{T}: & \begin{bmatrix} \text{T}: & X \end{bmatrix} \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \text{CAT}: & at \end{bmatrix} \quad \begin{bmatrix} \text{CAT}: & ap \\ \text{T}: & X \end{bmatrix}$$

$$\rho_3: \quad \begin{bmatrix} \text{CAT}: & ap \\ \text{T}: & end \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \text{CAT}: & at \end{bmatrix}$$

$$\rho_4: \quad \begin{bmatrix} \text{CAT}: & bp \\ \text{T}: & \begin{bmatrix} \text{T}: & X \end{bmatrix} \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \text{CAT}: & bt \end{bmatrix} \quad \begin{bmatrix} \text{CAT}: & bp \\ \text{T}: & X \end{bmatrix}$$

$$\rho_5: \quad \begin{bmatrix} \text{CAT}: & bp \\ \text{T}: & end \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \text{CAT}: & bt \end{bmatrix}$$

$$\rho_6: \quad \begin{bmatrix} \text{CAT}: & cp \\ \text{T}: & \begin{bmatrix} \text{T}: & X \end{bmatrix} \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \text{CAT}: & ct \end{bmatrix} \quad \begin{bmatrix} \text{CAT}: & cp \\ \text{T}: & X \end{bmatrix}$$

$$\rho_7: \quad \begin{bmatrix} \text{CAT}: & cp \\ \text{T}: & end \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \text{CAT}: & ct \end{bmatrix}$$

Example: (continued)

$$\begin{bmatrix}\text{CAT}: & at\end{bmatrix} \rightarrow a$$
$$\begin{bmatrix}\text{CAT}: & bt\end{bmatrix} \rightarrow b$$
$$\begin{bmatrix}\text{CAT}: & ct\end{bmatrix} \rightarrow c$$

Trans-context-free languages

Example: Derivation sequence of $a^2b^2c^2$
Start with a form that consists of the start symbol,

$$\sigma_0 = \begin{bmatrix}\text{CAT}: & s\end{bmatrix}.$$

Only one rule, ρ_1, can be applied to the single element of the multi-AVM in σ_0, yielding:

$$\sigma_1 = \begin{bmatrix}\text{CAT}: & ap\\ \text{T}: & X\end{bmatrix}\begin{bmatrix}\text{CAT}: & bp\\ \text{T}: & X\end{bmatrix}\begin{bmatrix}\text{CAT}: & cp\\ \text{T}: & X\end{bmatrix}$$

Example: (continued)
Applying ρ_2, to the first element of σ_1:

$$\sigma_2 = \begin{bmatrix}\text{CAT}: & at\end{bmatrix}\begin{bmatrix}\text{CAT}: & ap\\ \text{T}: & X\end{bmatrix}\begin{bmatrix}\text{CAT}: & bp\\ \text{T}: & \begin{bmatrix}\text{T}: & X\end{bmatrix}\end{bmatrix}\begin{bmatrix}\text{CAT}: & cp\\ \text{T}: & \begin{bmatrix}\text{T}: & X\end{bmatrix}\end{bmatrix}$$

We can now choose the third element in σ_2 and apply the rule ρ_4:

$$\sigma_3 = \begin{bmatrix}\text{CAT}: & at\end{bmatrix}\begin{bmatrix}\text{CAT}: & ap\\ \text{T}: & X\end{bmatrix}\begin{bmatrix}\text{CAT}: & bt\end{bmatrix}\begin{bmatrix}\text{CAT}: & bp\\ \text{T}: & X\end{bmatrix}\begin{bmatrix}\text{CAT}: & cp\\ \text{T}: & \begin{bmatrix}\text{T}: & X\end{bmatrix}\end{bmatrix}$$

Applying ρ_6 to the fifth element of $\sigma 3$, we get:

$$\sigma_4 = \begin{bmatrix}\text{CAT}: & at\end{bmatrix}\begin{bmatrix}\text{CAT}: & ap\\ \text{T}: & X\end{bmatrix}\begin{bmatrix}\text{CAT}: & bt\end{bmatrix}\begin{bmatrix}\text{CAT}: & bp\\ \text{T}: & X\end{bmatrix}\begin{bmatrix}\text{CAT}: & ct\end{bmatrix}\begin{bmatrix}\text{CAT}: & cp\\ \text{T}: & X\end{bmatrix}$$

Example: (continued)
The second element of σ_4 is unifiable with the heads of both ρ_2 and ρ_3. We choose to apply ρ_3:

$$\sigma_5 = \begin{bmatrix}\text{CAT}: & at\end{bmatrix}\begin{bmatrix}\text{CAT}: & at\end{bmatrix}\begin{bmatrix}\text{CAT}: & bt\end{bmatrix}\begin{bmatrix}\text{CAT}: & bp\\ \text{T}: & end\end{bmatrix}\begin{bmatrix}\text{CAT}: & ct\end{bmatrix}\begin{bmatrix}\text{CAT}: & cp\\ \text{T}: & end\end{bmatrix}$$

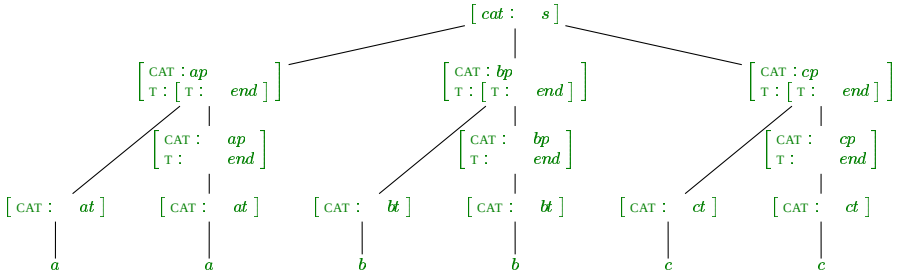
In the same way we can now apply ρ_5 and ρ_7 and obtain, eventually,

$$\sigma_7 = \begin{bmatrix}\text{CAT}: & at\end{bmatrix}\begin{bmatrix}\text{CAT}: & at\end{bmatrix}\begin{bmatrix}\text{CAT}: & bt\end{bmatrix}\begin{bmatrix}\text{CAT}: & bt\end{bmatrix}\begin{bmatrix}\text{CAT}: & ct\end{bmatrix}\begin{bmatrix}\text{CAT}: & ct\end{bmatrix}$$

Now, let $w = aabbcc$; then σ_7 is a member of $PT_w(1,6)$; in fact, it is the only member of the preterminal set. Therefore, $w \in L(G_{abc})$.

Trans-context-free languages

Example: Derivation tree of $a^2b^2c^2$



The repetition language

Example: A unification grammar for the language $\{ww \mid w \in \{a,b\}^+\}$

The signature of the grammar consists in the features CAT, FIRST and REST and the atoms *s*, *ap*, *bp*, *at*, *bt* and *elist*. The terminal symbols are *a* and *b*. The start symbol is the left-hand side of the first rule.

Example: (continued)

$$\left[\begin{array}{ll}\text{CAT}: & s\end{array}\right] \rightarrow \left[\begin{array}{ll}\text{FIRST}: & X\\ \text{REST}: & Y\end{array}\right] \left[\begin{array}{ll}\text{FIRST}: & X\\ \text{REST}: & Y\end{array}\right]$$

$$\left[\begin{array}{ll}\text{FIRST}: & ap\\ \text{REST}: & \left[\begin{array}{ll}\text{FIRST}: & X\\ \text{REST}: & Y\end{array}\right]\end{array}\right] \rightarrow \left[\begin{array}{ll}\text{CAT}: & at\end{array}\right] \left[\begin{array}{ll}\text{FIRST}: & X\\ \text{REST}: & Y\end{array}\right]$$

$$\left[\begin{array}{ll}\text{FIRST}: & bp\\ \text{REST}: & \left[\begin{array}{ll}\text{FIRST}: & X\\ \text{REST}: & Y\end{array}\right]\end{array}\right] \rightarrow \left[\begin{array}{ll}\text{CAT}: & bt\end{array}\right] \left[\begin{array}{ll}\text{FIRST}: & X\\ \text{REST}: & Y\end{array}\right]$$

$$\left[\begin{array}{ll}\text{FIRST}: & ap\\ \text{REST}: & elist\end{array}\right] \rightarrow \left[\begin{array}{ll}\text{CAT}: & at\end{array}\right]$$

$$\left[\begin{array}{ll}\text{FIRST}: & bp\\ \text{REST}: & elist\end{array}\right] \rightarrow \left[\begin{array}{ll}\text{CAT}: & bt\end{array}\right]$$

$$\left[\begin{array}{ll}\text{CAT}: & at\end{array}\right] \rightarrow a$$

$$\left[\begin{array}{ll}\text{CAT}: & bt\end{array}\right] \rightarrow b$$

Unification grammars and Turing machines

Unification grammars can simulate the operation of Turing machines.

The membership problem for unification grammars is as hard as the halting problem.

Unification grammars and Turing machines

A (deterministic) **Turing machine** $(Q, \Sigma, \flat, \delta, s, h)$ is a tuple such that:

- Q is a finite set of states
- Σ is an alphabet, not containing the symbols L, R and $elist$
- $\flat \in \Sigma$ is the blank symbol
- $s \in Q$ is the initial state
- $h \in Q$ is the final state
- $\delta : (Q \backslash \{h\}) \times \Sigma \to Q \times (\Sigma \cup \{L, R\})$ is a total function specifying transitions.

Unification grammars and Turing machines

A **configuration** of a Turing machine consists of the state, the contents of the tape and the position of the head on the tape.

A configuration is depicted as a quadruple (q, w_l, σ, w_r) where $q \in Q$, $w_l, w_r \in \Sigma^*$ and $\sigma \in \Sigma$; in this case, the contents of the tape is $\flat^\omega \cdot w_l \cdot \sigma \cdot w_r \cdot \flat^\omega$, and the head is positioned on the σ symbol.

A given configuration yields a next configuration, determined by the transition function δ, the current state and the character on the tape that the head points to.

Unification grammars and Turing machines

Let

$$\text{first}(\sigma_1 \cdots \sigma_n) = \begin{cases} \sigma_1 & n > 0 \\ \flat & n = 0 \end{cases} \qquad \text{but-first}(\sigma_1 \cdots \sigma_n) = \begin{cases} \sigma_2 \cdots \sigma_n & n > 1 \\ \epsilon & n \leq 1 \end{cases}$$

$$\text{last}(\sigma_1 \cdots \sigma_n) = \begin{cases} \sigma_n & n > 0 \\ \flat & n = 0 \end{cases} \qquad \text{but-last}(\sigma_1 \cdots \sigma_n) = \begin{cases} \sigma_1 \cdots \sigma_{n-1} & n > 1 \\ \epsilon & n \leq 1 \end{cases}$$

Unification grammars and Turing machines

Then the next configuration of a configuration (q, w_l, σ, w_r) is defined iff $q \neq h$, in which case it is:

(p, w_l, σ', w_r)	if $\delta(q, \sigma) = (p, \sigma')$ where $\sigma' \in \Sigma$
$(p, w_l\sigma, \text{first}(w_r), \text{but-first}(w_r))$	if $\delta(q, \sigma) = (p, R)$
$(p, \text{but-last}(w_l), \text{last}(w_l), \sigma w_r)$	if $\delta(q, \sigma) = (p, L)$

Unification grammars and Turing machines

A next configuration is only defined for configurations in which the state is not the final state, h.

Since δ is a total function, there always exists a unique next configuration for every given configuration.

We say that a configuration c_1 yields the configuration c_2, denoted $c_1 \vdash c_2$, iff c_2 is the next configuration of c_1.

Unification grammars and Turing machines

Program:

- define a unification grammar G_M for every Turing machine M such that the grammar generates the word halt if and only if the machine accepts the empty input string:

$$L(G_M) = \begin{cases} \{\text{halt}\} & \text{if } M \text{ terminates for the empty input} \\ \emptyset & \text{if } M \text{ does not terminate on the empty input} \end{cases}$$

- if there were a decision procedure to determine whether $w \in L(G)$ for an arbitrary unification grammar G, then in particular such a procedure could determine membership in the language of G_M, simulating the Turing machine M.

- the procedure for deciding whether $w \in L(G)$, when applied to

the problem halt$\in L(G_M)$, determines whether M terminates for the empty input, which is known to be undecidable.

Unification grammars and Turing machines

Feature structures will have three features: curr, representing the character under the head; right, representing the tape contents to the right of the head (as a list); and left, representing the tape contents to the left of the head, in a reversed order.

All the rules in the grammar are unit rules; and the only terminal symbol is halt. Therefore, the language generated by the grammar is necessarily either the singleton {halt} or the empty set.

Unification grammars and Turing machines: signature

Let $M = (Q, \Sigma, \flat, \delta, s, h)$ be a Turing machine. Define a unification grammar G_M as follows:

- FEATS $= \{$CAT, LEFT, RIGHT, CURR, FIRST, REST$\}$
- ATOMS $= \Sigma \cup \{$*start*, *elist*$\}$.
- The start symbol is $\begin{bmatrix}\text{CAT}: & start\end{bmatrix}$.
- the only terminal symbol is halt.

Unification grammars and Turing machines: rules

Two rules are defined for every Turing machine:

$$\begin{bmatrix}\text{CAT}: & start\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & s\\ \text{CURR}: & \flat\\ \text{RIGHT}: & elist\\ \text{LEFT}: & elist\end{bmatrix}$$

$$h \rightarrow \text{halt}$$

Unification grammars and Turing machines: rules

For every q, σ such that $\delta(q, \sigma) = (p, \sigma')$ and $\sigma' \in \Sigma$, the following rule is defined:

$$\begin{bmatrix}\text{CAT}: & q\\ \text{CURR}: & \sigma\\ \text{RIGHT}: & X\\ \text{LEFT}: & Y\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & p\\ \text{CURR}: & \sigma'\\ \text{RIGHT}: & X\\ \text{LEFT}: & Y\end{bmatrix}$$

Unification grammars and Turing machines: rules

For every q, σ such that $\delta(q, \sigma) = (p, R)$ we define two rules:

$$\begin{bmatrix}\text{CAT}: & q\\ \text{CURR}: & \sigma\\ \text{RIGHT}: & elist\\ \text{LEFT}: & X\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & p\\ \text{CURR}: & \flat\\ \text{RIGHT}: & elist\\ \text{LEFT}: & \begin{bmatrix}\text{FIRST}: & \sigma\\ \text{REST}: & X\end{bmatrix}\end{bmatrix}$$

$$\begin{bmatrix}\text{CAT}: & q\\ \text{CURR}: & \sigma\\ \text{RIGHT}: & \begin{bmatrix}\text{FIRST}: & X\\ \text{REST}: & Y\end{bmatrix}\\ \text{LEFT}: & W\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & p\\ \text{CURR}: & X\\ \text{RIGHT}: & Y\\ \text{LEFT}: & \begin{bmatrix}\text{FIRST}: & \sigma\\ \text{REST}: & W\end{bmatrix}\end{bmatrix}$$

Unification grammars and Turing machines: rules

For every q, σ such that $\delta(q, \sigma) = (p, L)$ we define two rules:

$$\begin{bmatrix} \text{CAT}: & q \\ \text{CURR}: & \sigma \\ \text{RIGHT}: & X \\ \text{LEFT}: & \mathit{elist} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{CURR}: & \flat \\ \text{RIGHT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & X \end{bmatrix} \\ \text{LEFT}: & \mathit{elist} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & q \\ \text{CURR}: & \sigma \\ \text{RIGHT}: & X \\ \text{LEFT}: & \begin{bmatrix} \text{FIRST}: & Y \\ \text{REST}: & W \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{CURR}: & Y \\ \text{RIGHT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & X \end{bmatrix} \\ \text{LEFT}: & W \end{bmatrix}$$

Unification grammars and Turing machines: results

Lemma 1. *Let c_1, c_2 be configurations of a Turing machine M, and A_1, A_2 be AVMs encoding these configurations, viewed as multi-AVMs of length 1. Then $c_1 \vdash c_2$ iff $A_1 \Rightarrow A_2$ in G_m.*

Theorem 2. *A Turing machine M halts for the empty input iff halt $\in L(G_M)$.*

Corollary 3. *The universal recognition problem for unification grammars is undecidable.*