עיבוד שפות טבעיות

שולי וינטנר

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

Problems of G_0

Over-generation (agreement constraints are not imposed):

*Rachel feed the sheep

*The shepherds feeds the sheep

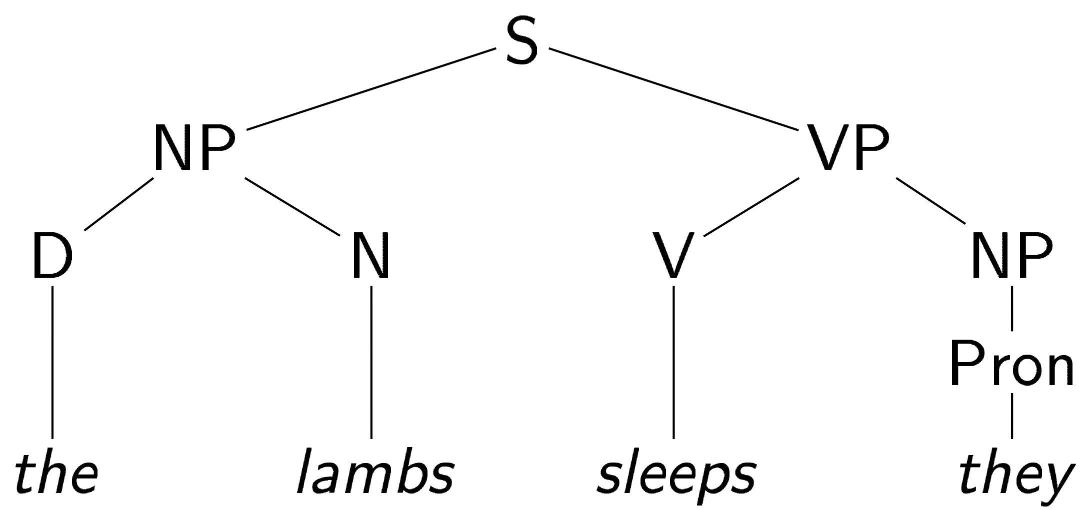
*Rachel feeds

*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.

An extended context-free formalism

Motivation: the violations of G_0: imposing on a grammar for E_0 person and number agreement constraints.

Basic idea: extend the CFG formalism with additional mechanisms, based on *feature structures*.

Augment the terminal symbols of a grammar, then generalize phrases and rules similarly.

The same techniques will be used to impose other constraints on a grammar for E_0.

Appropriate for dealing with various phenomena of natural languages, such as long-distance dependencies.

Preserve the context-free backbone of grammars.

An extended context-free formalism

The core idea is to incorporate into the grammar *properties* of symbols, in terms of which the violations of G_0 were stated.

Properties are represented by means of *feature structures*.

Such structures map *features* into *values* (themselves feature structures).

A special case of feature structures are *atoms*, which represent structureless values.

For example, to deal with *number*, we use a feature NUM, and a set of atomic feature structures {*sg*,*pl*} as its values.

A feature can also have an *unspecified value*, represented as an empty feature structure.

Associating feature structures with words

Attribute-value matrices (AVMs)

Each 'row' in an AVM is a pair $\textsc{f} : v$

Values can either be atomic or complex, in the form of another AVM

lamb

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

lambs

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

sheep

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

I

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

dreams

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

Associating feature structures with words

lambs

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

How to group features?

Variables

Two notations for variables:

$$\begin{bmatrix} \text{NUM}: & X([\,]) \\ \text{PERS}: & second \end{bmatrix} \qquad \begin{bmatrix} \text{NUM}: & \boxed{1}\,[\,] \\ \text{PERS}: & second \end{bmatrix}$$

Value sharing (reentrancy):

$$\begin{bmatrix} \text{F}: & X(a) \\ \text{G}: & X(a) \end{bmatrix} \qquad \begin{bmatrix} \text{F}: & \boxed{1}\,a \\ \text{G}: & \boxed{1}\,a \end{bmatrix}$$

Attribute-value matrices

An AVM is a *syntactic* object, consisting of 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 can be either *atoms* (drawn from a fixed set ATOMS), or, recursively, AVMs themselves.

Some AVMs are assigned *variables*.

Two occurrences of the same variable (within the scope of that variable) denote one and the *same* value.

FEATS and ATOMS are parameters for the collection of AVMs, and are referred to as the *signature*.

Attribute-value matrices

$$\begin{bmatrix} \text{SUBJ}: & \begin{bmatrix} \text{AGR}: & \boxed{1}\begin{bmatrix} \text{NUM}: & sg \\ \text{PERS}: & 3rd \end{bmatrix} \\ \text{CASE}: & nom \end{bmatrix} \\ \text{PRED}: & \begin{bmatrix} \text{AGR}: & \boxed{1} \\ \text{TENSE}: & past \end{bmatrix} \end{bmatrix}$$

Attribute-value matrices

If

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

is a feature structure then the *domain* of A is $dom(A) = \{\mathrm{F}_i \mid 1 \leq i \leq n\}$.

$|dom(A)| = n$ so for every $1 \leq i, j \leq n$ such that $i \neq j$, $\mathrm{F}_i \neq f_j$.

The *empty* AVM, denoted $[\,]$, has as its domain the empty set of features.

Attribute-value matrices

Some AVMs can be assigned *variables*.

An AVM is *well-formed* if for every variable occurring in it, all its occurrences are associated with the same value.

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

not well-formed
$$\begin{bmatrix} \text{F}: & \boxed{1} \begin{bmatrix} \text{H}: & a \end{bmatrix} \\ \text{G}: & \boxed{1}\, b \end{bmatrix}$$

not well-formed
$$\begin{bmatrix} \text{F}: & X\left(\begin{bmatrix} \text{H}: & a \end{bmatrix}\right) \\ \text{G}: & X\left(\begin{bmatrix} \text{K}: & b \end{bmatrix}\right) \end{bmatrix}$$

Attribute-value matrices

Conventions:

- since multiple occurrences of the same variables always are associated with the same values, only one instance of this value is explicated

- whenever a variable is associated with the empty AVM, the AVM itself is omitted.

Attribute-value matrices

The *value* of a feature F_i in A is $val(A, \text{F}_i) = A_i$.

If $\text{F} \notin dom(A)$ then $val(A, \text{F})$ is undefined.

A *path* is a (possibly empty) sequence of features that can be used to pick a value in a feature structure.

The notion of values is extended from features to paths: $val(A, \pi)$ is the value associated with 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 non-atomic feature structure A.

Attribute-value matrices

Let

$$A = \begin{bmatrix} \textsc{agr}: & \begin{bmatrix} \textsc{num}: & pl \\ \textsc{pers}: & third \end{bmatrix} \end{bmatrix}$$

Then

$$dom(A) = \{\textsc{agr}\}, val(A, \textsc{agr}) = \begin{bmatrix} \textsc{num}: & pl \\ \textsc{pers}: & third \end{bmatrix}.$$

The paths of A are $\{\epsilon, \langle\textsc{agr}\rangle, \langle\textsc{agr,num}\rangle, \langle\textsc{agr,pers}\rangle\}$.

The values of these paths are: $val(A, \epsilon) = A$, $val(A, \langle\textsc{agr,num}\rangle) = pl$, $val(A, \langle\textsc{agr,pers}\rangle) = third$. $val(A, \langle\textsc{num,agr}\rangle)$ is undefined.

Attribute-value matrices

val is a non-compositional notion:

$$A = \begin{bmatrix} \mathrm{F}: & X \end{bmatrix}, \quad B = \begin{bmatrix} \mathrm{F}: & Y \end{bmatrix}$$

$$val(A, \mathrm{F}) = val(B, \mathrm{F}) = [\,]$$

$$C = \begin{bmatrix} \mathrm{F}: & X \\ \mathrm{G}: & X(a) \end{bmatrix}, \quad D = \begin{bmatrix} \mathrm{F}: & Y \\ \mathrm{G}: & X(a) \end{bmatrix}$$

$$val(C, \mathrm{F}) = a \neq [\,] = val(D, \mathrm{F})$$

Reentrancy

The difference between *type*- and *token*- identity:

$$(1) \quad \begin{bmatrix} \mathrm{F}: & a \\ \mathrm{G}: & a \end{bmatrix} \qquad (2) \quad \begin{bmatrix} \mathrm{F}: & X(a) \\ \mathrm{G}: & X(a) \end{bmatrix}$$

Reentrancy

$$A_1 = \begin{bmatrix} \text{F}_1 : & \begin{bmatrix} \text{G} : & a \end{bmatrix} \\ \text{F}_2 : & \begin{bmatrix} \text{G} : & a \end{bmatrix} \end{bmatrix}, \quad A_2 = \begin{bmatrix} \text{F}_1 : & \boxed{1} \begin{bmatrix} \text{G} : & a \end{bmatrix} \\ \text{F}_2 : & \boxed{1} \end{bmatrix}$$

$val(A_1, \text{F}_1) = val(A_1, \text{F}_2) = val(A_2, \text{F}_1) = val(A_2, \text{F}_2) = \begin{bmatrix} \text{G} : & a \end{bmatrix}$.

Suppose that by some action, a feature H with the value b is added to the value of F_1 in both A_1 and A_2:

$$A_1' = \begin{bmatrix} \text{F}_1 : & \begin{bmatrix} \text{G} : & a \\ \text{H} : & b \end{bmatrix} \\ \text{F}_2 : & \begin{bmatrix} \text{G} : & a \end{bmatrix} \end{bmatrix}, \quad A_2' = \begin{bmatrix} \text{F}_1 : & \boxed{1} \begin{bmatrix} \text{G} : & a \\ \text{H} : & b \end{bmatrix} \\ \text{F}_2 : & \boxed{1} \end{bmatrix}$$

Reentrancy

The notion of reentrancy is extended to paths: two paths are said to be reentrant if they are associated with the same (token-identical) value.

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

$\langle \mathrm{F}_1, \mathrm{G} \rangle$ and $\langle \mathrm{F}_2, \mathrm{G} \rangle$ are reentrant, implying $val(A_3, \langle \mathrm{F}_1, \mathrm{G} \rangle) = val(A_3, \langle \mathrm{F}_2, \mathrm{G} \rangle) = \begin{bmatrix} \mathrm{H} : & b \end{bmatrix}$.

Cycles

A special case of reentrancy is *cyclicity*:

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

Subsumption

Feature structures are used to represent information.

The amount of information stored within different feature structures can be compared, thus inducing a natural (partial) pre-order on the structures.

This relation is called *subsumption* and is denoted '$\sqsubseteq$'.

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 atom then B is an identical atom;
2. for every $\textsc{f} \in \textsc{Feats}$, if $\textsc{f} \in dom(A)$ then $\textsc{f} \in dom(B)$, and $val(A, \textsc{f})$ subsumes $val(B, \textsc{f})$; and
3. if two paths are reentrant in A, they are also reentrant in B.

Subsumption

The empty feature structure, $[\,]$, is the most general feature structure, subsuming all others, including atoms (as it encodes no information at all):

$$[\,] \sqsubseteq \left[\textsc{num}: \quad sg\right]$$

In the same way,

$$\left[\textsc{num}: \quad X\right] \sqsubseteq \left[\textsc{num}: \quad sg\right]$$

since by convention, X is a shorthand for $X\,[\,]$.

Subsumption

Adding more information results in a more specific feature structure:

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

Another way to add information is through reentrancies:

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

Subsumption

Subsumption is a *partial* pre-order: not every pair of feature structures is comparable:

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

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

$$\left[\textsc{num}: \; sg\right] \begin{array}{c} \not\sqsubseteq \\ \not\sqsupseteq \end{array} \left[\textsc{pers}: \; third\right]$$

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 $A \sqsubseteq C$.

Feature structure equality

When are two atomic feature structures equal?

Two atomic feature structures, bearing one and the same atom, are not necessarily identical. Of course, it is possible to require such identity by associating the two feature structures with the same variable.

Subsumption is not anti-symmetric: it is possible for two feature structures to mutually subsume each other, while not being identical.

Feature structure equality

Feature structures are intensional objects: two AVMs are not separable by paths:

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

This leads naturally to their mathematical representation as graphs.

Unification

Unification, denoted '$\sqcup$', is an information combination operation. It 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*). Otherwise, the unification *fails*, and the operands are said to be *inconsistent*. Failure is denote by $\top$.

In terms of the subsumption ordering, non-failing unification returns the least upper bound (*lub*) of its arguments.

Unification: examples

Unification combines consistent information:

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

Unification: examples

Different atoms are inconsistent:

$$\begin{bmatrix}\textsc{num}: & sg\end{bmatrix} \sqcup \begin{bmatrix}\textsc{num}: & pl\end{bmatrix} = \top$$

Unification: examples

Atoms and non-atoms are inconsistent:

$$\begin{bmatrix}\text{NUM}: & sg\end{bmatrix} \sqcup sg = \top$$

Unification: examples

Unification is absorbing:

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

Unification: examples

Empty feature structures are identity elements:

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

Unification: examples

Reentrancy causes two consistent values to coincide:

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

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

Unification: examples

Variables can be (partially) instantiated:

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

...but they are not lost, and can be used for further instantiation:

$$\begin{bmatrix}\mathrm{F}: & X\end{bmatrix} \sqcup \begin{bmatrix}\mathrm{F}: & a\end{bmatrix} \sqcup \begin{bmatrix}\mathrm{G}: & X\end{bmatrix} =$$

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

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

Unification: examples

Unification acts differently depending on whether the values are equal:

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

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

Unification: examples

…or identical:

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

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

Unification and variables

Unification *binds* variables together. When two feature structures are unified, and each of them is associated with some variable, then the two variables are said to be bound to each other, which means that they both share one and the same value (the result of the unification).

The *scope* of the variables must be taken into account: all other occurrences of the same variables in this scope are effected.

Unification and variables

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

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

Since the variables $\boxed{1}$ and $\boxed{2}$ occur nowhere else,

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

Unification and variables

However, had either $\boxed{1}$ or $\boxed{2}$ occurred elsewhere, their values would have been modified as a result of the unification:

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

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

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 (or *anti-unification*) is the inverse of unification.

Defined over pairs of feature structures, generalization (denoted $\sqcap$) is the operation that returns the most specific (or least general) feature structure that is still more general than both arguments.

In terms of the subsumption ordering, generalization is the greatest lower bound (*glb*) of two feature structures.

Unlike unification, generalization can never fail.

Generalization

Generalization reduces information:

$$\left[\text{NUM}: \quad sg\right] \sqcap \left[\text{PERS}: \quad third\right] = [\,]$$

Generalization

Different atoms are inconsistent:

$$\left[\text{NUM}: \quad sg\right] \sqcap \left[\text{NUM}: \quad pl\right] = [\,]$$

Generalization

Generalization is restricting:

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

Generalization

Empty feature structures are zero elements:

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

Generalization

Reentrancies can be lost:

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

$$\begin{bmatrix} \text{F}: & \begin{bmatrix} \text{NUM}: & sg \end{bmatrix} \\ \text{G}: & \begin{bmatrix} \text{NUM}: & sg \end{bmatrix} \end{bmatrix}$$

Using feature structures for representing lists

Feature structures can be easily used to encode (finite) lists.

A list can be simply represented as a feature structure having two features, named, say, FIRST and REST.

The value of the FIRST feature is the first element of the list; the value of REST is either the atom *elist*, denoting an empty list, or itself a list.

Using feature structures for representing lists

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

List traversal is analogous to computing the value of a path.

A feature structure encoding of lists is a representation method only; it does not provide operations on lists.

Using feature structures for representing lists

An encoding of the list $\langle\langle 1,2,6,7\rangle,\langle 3,4\rangle,\langle 5\rangle\rangle$:

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

Using feature structures for representing graphs

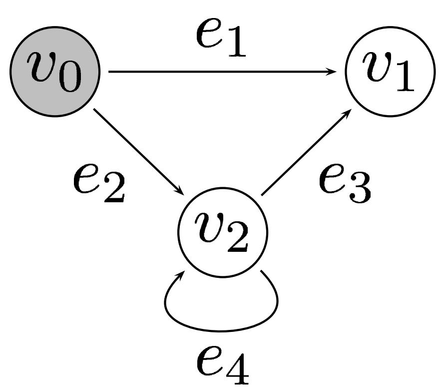
Let $G = (V, E)$ be a directed, rooted graph, where V is a set of vertices and E is a set of edges (a subset of $V \times V$).

G can be encoded as a feature structure by associating an empty feature structure A_v with each node $v \in V$.

Then, whenever an edge $e \in E$ leads from v to u, a feature F_e is defined in A_v, with A_u as its value.

The encoding of G is obtained by considering the feature structure A_q associated with the root q of G.

Using feature structures for representing graphs



$$\begin{bmatrix} \mathrm{F}_{e_1} : & \boxed{1}\,[\,] \\ \mathrm{F}_{e_2} : & \boxed{2} \begin{bmatrix} \mathrm{F}_{e_3} : & \boxed{1} \\ \mathrm{F}_{e_4} : & \boxed{2} \end{bmatrix} \end{bmatrix}$$

Adding features to rules

Phrases, too, 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.

$$\mathit{NP}\begin{bmatrix}\mathrm{NUM}: & [\,] \\ \mathrm{PERS}: & [\,]\end{bmatrix}$$

Adding features to rules

Generalized categories (or *extended* ones) have a *base category* and an associated feature structure.

Productions are endowed with structural information by assigning a feature structure to every non-terminal symbol in the CFG skeleton of the production.

Adding features to rules

Imposing number agreement in E_0:

$$(1) \quad \underset{[\text{NUM}: \; X]}{N} \rightarrow \underset{[\text{NUM}: \; X(sg)]}{\textit{lamb}}$$

$$(2) \quad \underset{[\text{NUM}: \; X]}{N} \rightarrow \underset{[\text{NUM}: \; X(pl)]}{\textit{lambs}}$$

$$(3) \quad S \rightarrow \underset{[\text{NUM}: \; X]}{NP} \; \underset{[\text{NUM}: \; X]}{VP}$$

$$(4) \quad \underset{[\text{NUM}: \; X]}{NP} \rightarrow \underset{[\text{NUM}: \; X]}{D} \; \underset{[\text{NUM}: \; X]}{N}$$

Adding features to rules

Scope of variables

$$\underset{\left[\textsc{num}:\ \ Y\right]}{N} \rightarrow \underset{\left[\textsc{num}:\ \ Y(pl)\right]}{\textit{lambs}}$$

Agreement as a declarative constraint

The undirectional view of agreement as a typical constraint-satisfaction view in unification grammars

Multi-AVMs

Sequences of AVMs, where some sub-structures can be shared among two or more AVMSs.

The scope of variables is extended from a single AVM to a multi-AVM.

The *length* of a multi-AVM is the number of it elements.

$$\sigma = \begin{bmatrix} \text{F}: & \begin{bmatrix} \text{G}: & a \\ \text{H}: & X \end{bmatrix} \end{bmatrix} \begin{bmatrix} \text{G}: & Y \end{bmatrix} \begin{bmatrix} \text{F}: & \begin{bmatrix} \text{H}: & b \\ \text{G}: & X \end{bmatrix} \\ \text{H}: & a \end{bmatrix}$$

$$val(\sigma, 1, \langle \text{F} \rangle) = \begin{bmatrix} \text{G}: & a \\ \text{H}: & [\,] \end{bmatrix} val(\sigma, 3, \langle \text{F} \rangle) = \begin{bmatrix} \text{H}: & b \\ \text{G}: & [\,] \end{bmatrix}$$

G_1, a unification grammar

$$S \rightarrow \underset{[\mathrm{NUM}:\ X]}{NP} \quad \underset{[\mathrm{NUM}:\ X]}{VP}$$

$$\underset{[\mathrm{NUM}:\ X]}{NP} \rightarrow \underset{[\mathrm{NUM}:\ X]}{D} \quad \underset{[\mathrm{NUM}:\ X]}{N}$$

$$\underset{[\mathrm{NUM}:\ X]}{VP} \rightarrow \underset{[\mathrm{NUM}:\ X]}{V}$$

$$\underset{[\mathrm{NUM}:\ X]}{VP} \rightarrow \underset{[\mathrm{NUM}:\ X]}{V} \quad \underset{[\mathrm{NUM}:\ Y]}{NP}$$

G_1, a unification grammar

$$\begin{array}{ccccc} \textit{N} & & \textit{lamb} & & \textit{sheep} \\ [\text{NUM}: \ X] & \to & [\text{NUM}: \ X(sg)] & | & [\text{NUM}: \ X] \\ \textit{V} & & \textit{sleeps} & & \textit{sleep} \\ [\text{NUM}: \ X] & \to & [\text{NUM}: \ X(sg)] & | & [\text{NUM}: \ X(pl)] \\ \textit{D} & & \textit{a} & & \textit{two} \\ [\text{NUM}: \ X] & \to & [\text{NUM}: \ X(sg)] & | & [\text{NUM}: \ X(pl)] \end{array}$$

Unification grammars

- Forms (and sentential forms)
- Rule application
- Derivations
- Languages

Unification grammars

A *form* is a sequence of base categories augmented by a multi-AVM of the same length:

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

Derivation is a binary relation over generalized forms.

Unification grammars

If α is a generalized form and $B_0 \to B_1 B_2 \ldots B_k$ is a grammar rule, application of the rule to the form 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.

Unification grammars

Matching: equality of the base categories; consistency of the feature structures

Replacing: unification in context

Matching

Suppose that

$$\alpha = \underset{[\text{NUM}: \; Y]}{\mathit{NP}} \; \underset{[\text{NUM}: \; Y]}{\mathit{VP}}$$

is a (sentential) form and that

$$\rho = \underset{[\text{NUM}: \; X]}{\mathit{VP}} \rightarrow \underset{[\text{NUM}: \; X]}{\mathit{V}} \; \underset{[\text{NUM}: \; W]}{\mathit{NP}}$$

is a rule.

The selected category matches the head of the rule ρ.

Replacing

The selected element of the form and the head of the rule 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.

Derivation step

As in the previous example, let

$$\alpha = \underset{[\textsc{num}:\ Y]}{NP}\ \underset{[\textsc{num}:\ Y]}{VP}$$

$$\rho = \underset{[\textsc{num}:\ X]}{VP} \rightarrow \underset{[\textsc{num}:\ X]}{V}\ \underset{[\textsc{num}:\ W]}{NP}$$

be a form and a rule, respectively. Applying the rule ρ to the form α results in a new sentential form, β, (in which the variables were renamed):

$$\beta = \underset{[\textsc{num}:\ X1]}{NP}\ \underset{[\textsc{num}:\ X1]}{V}\ \underset{[\textsc{num}:\ W1]}{NP}$$

Derivation step

Now assume that the (terminal) rule

$$\underset{[\textsc{num}:\ Y]}{\textit{V}} \rightarrow \underset{[\textsc{num}:\ Y(sg)]}{\textit{herds}}$$

is to be applied to β:

$$\beta = \underset{[\textsc{num}:\ X1]}{\textit{NP}} \quad \underset{[\textsc{num}:\ X1]}{\textit{V}} \quad \underset{[\textsc{num}:\ W1]}{\textit{NP}}$$

The result is:

$$\gamma = \underset{[\textsc{num}:\ Z2]}{\textit{NP}} \quad \underset{[\textsc{num}:\ Z2(sg)]}{\textit{herds}} \quad \underset{[\textsc{num}:\ W2]}{\textit{NP}}$$

Derivation step

The rule itself may change during unification:

$$\underset{\left[\text{NUM}:\ X\right]}{\textit{NP}} \rightarrow \underset{\left[\text{NUM}:\ X\right]}{\textit{D}}\ \underset{\left[\text{NUM}:\ X\right]}{\textit{N}}$$

$$\gamma = \underset{\left[\text{NUM}:\ Z2\right]}{\textit{NP}}\ \underset{\left[\text{NUM}:\ Z2(sg)\right]}{\textit{herds}}\ \underset{\left[\text{NUM}:\ W2\right]}{\textit{NP}}$$

The result:

$$\underset{\left[\text{NUM}:\ Z3\right]}{\textit{D}}\ \underset{\left[\text{NUM}:\ Z3\right]}{\textit{N}}\ \underset{\left[\text{NUM}:\ Z3(sg)\right]}{\textit{herds}}\ \underset{\left[\text{NUM}:\ W3\right]}{\textit{NP}}$$

Derivation with ϵ-rules

Applying

$$\rho = \underset{\begin{bmatrix} \text{F}: & Z \\ \text{G}: & Z \end{bmatrix}}{B} \rightarrow \epsilon$$

to

$$\alpha = \underset{\begin{bmatrix} \text{F}: & X \end{bmatrix}}{A} \quad \underset{\begin{bmatrix} \text{F}: & X \\ \text{G}: & Y \end{bmatrix}}{B} \quad \underset{\begin{bmatrix} \text{G}: & Y \end{bmatrix}}{C}$$

yields:

$$\underset{\begin{bmatrix} \text{F}: & W \end{bmatrix}}{A} \quad \underset{\begin{bmatrix} \text{G}: & W \end{bmatrix}}{C}$$

Derivation

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

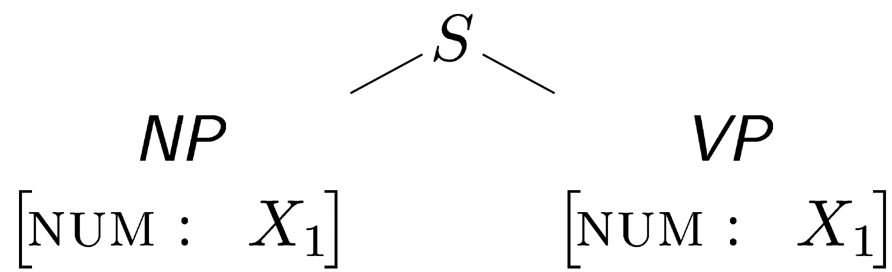
$$\Rightarrow \underset{[\text{NUM}:\ X_2]}{D} \quad \underset{[\text{NUM}:\ X_2]}{N} \quad \underset{[\text{NUM}:\ X_1]}{VP}$$

$$\Rightarrow \underset{[\text{NUM}:\ X_2]}{D} \quad \underset{[\text{NUM}:\ X_2]}{N} \quad \underset{[\text{NUM}:\ X_3]}{V}$$

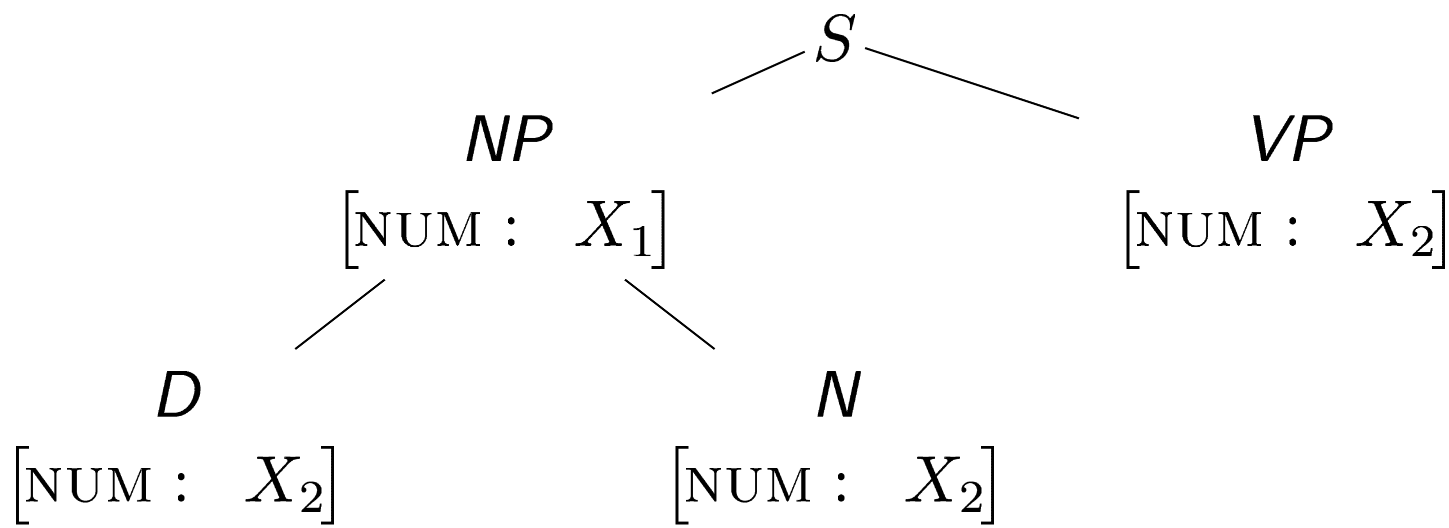
$$\Rightarrow \underset{[num:\ X_2(pl)]}{two} \quad \underset{[\text{NUM}:\ X_2]}{N} \quad \underset{[\text{NUM}:\ X_3]}{V}$$

$$\stackrel{*}{\Rightarrow} \underset{[\text{NUM}:\ X_2(pl)]}{two} \quad \underset{[\text{NUM}:\ X_2(pl)]}{sheep} \quad \underset{[\text{NUM}:\ X_3(pl)]}{sleep}$$

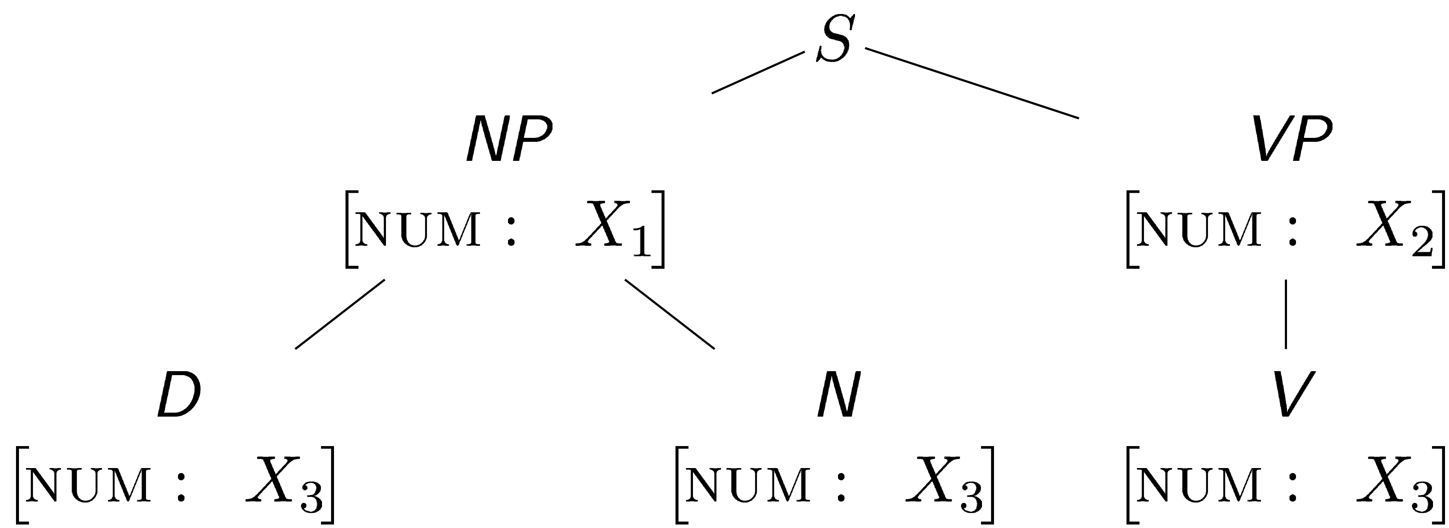
Derivation tree



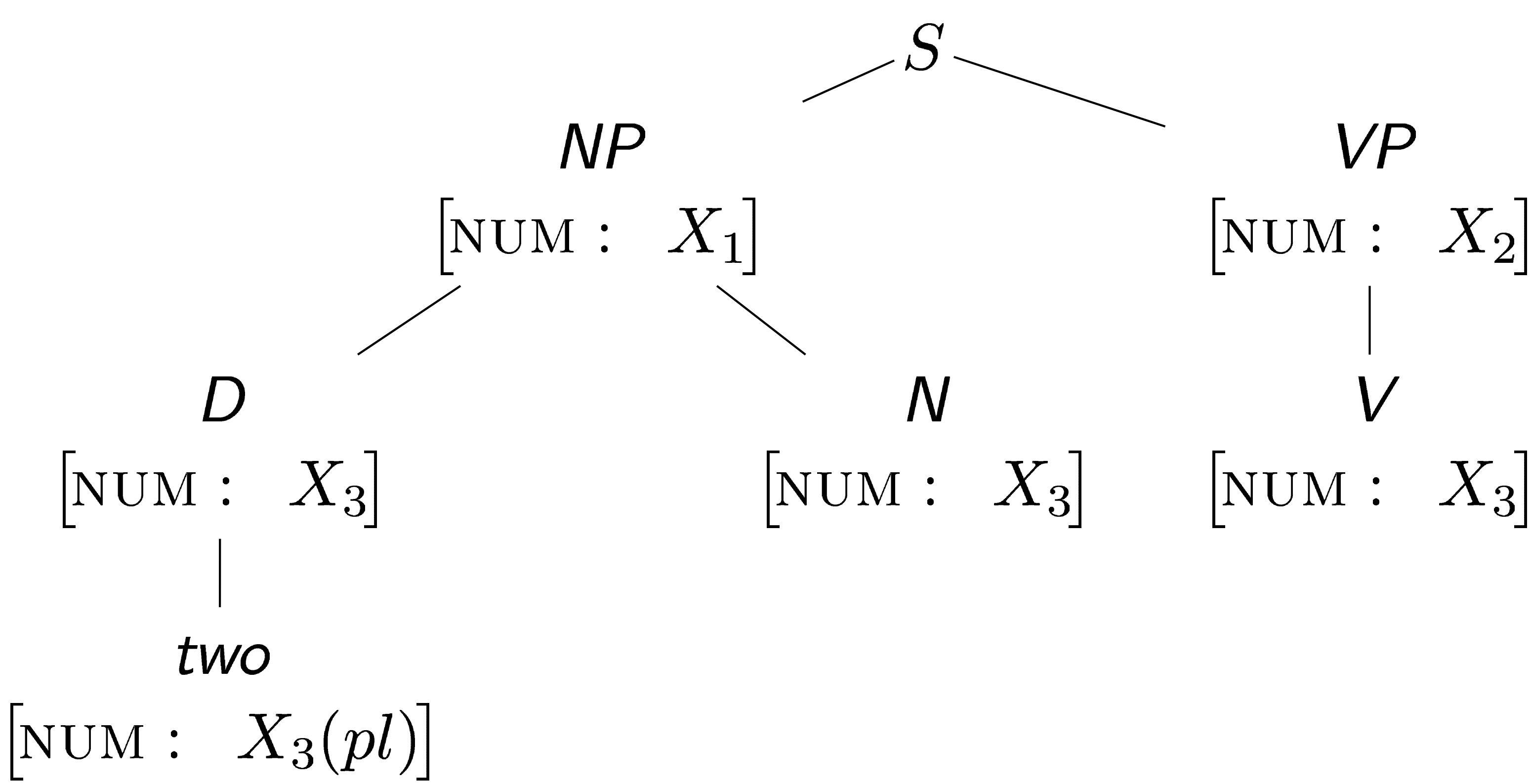
Derivation tree



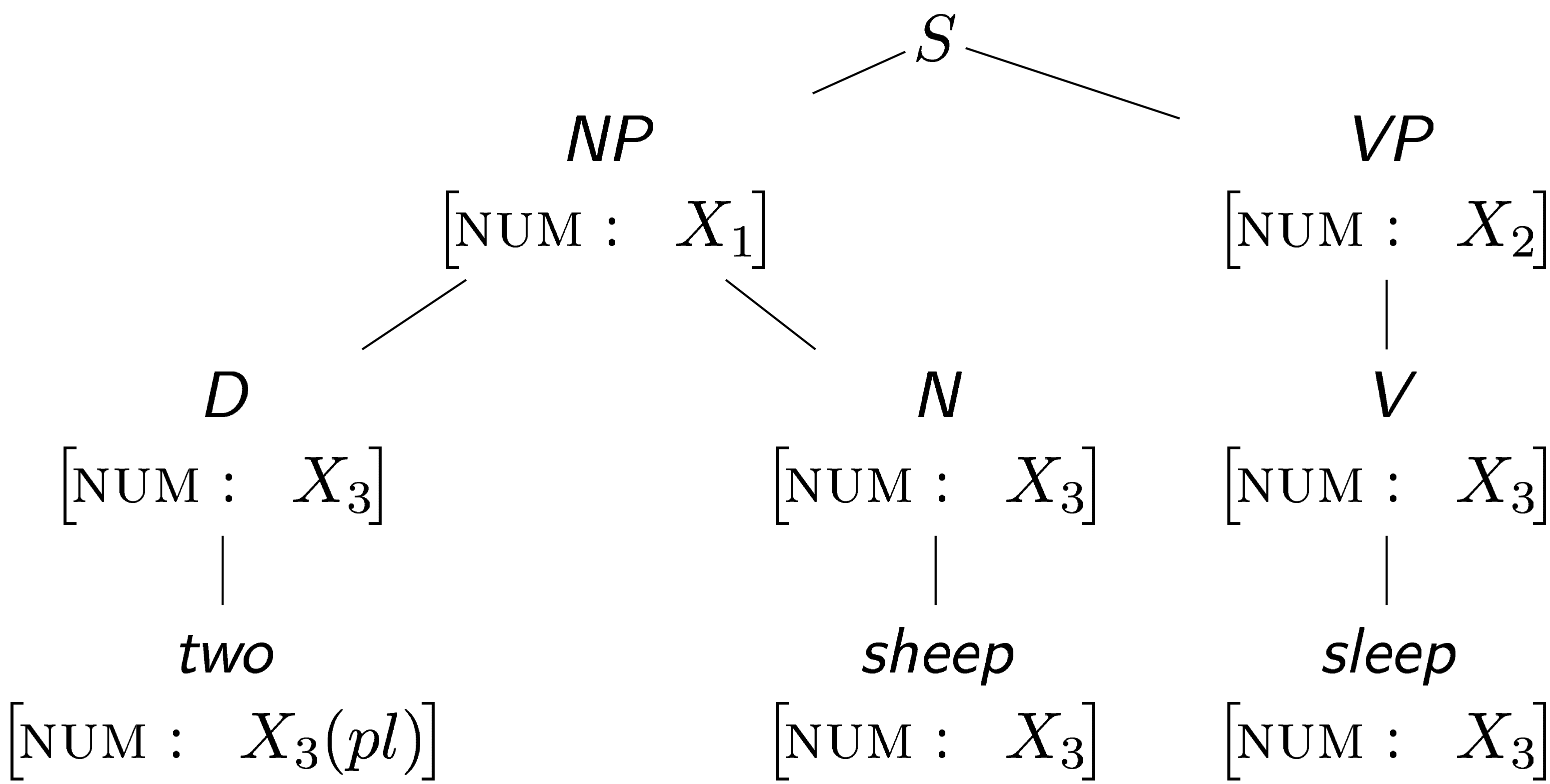
Derivation tree



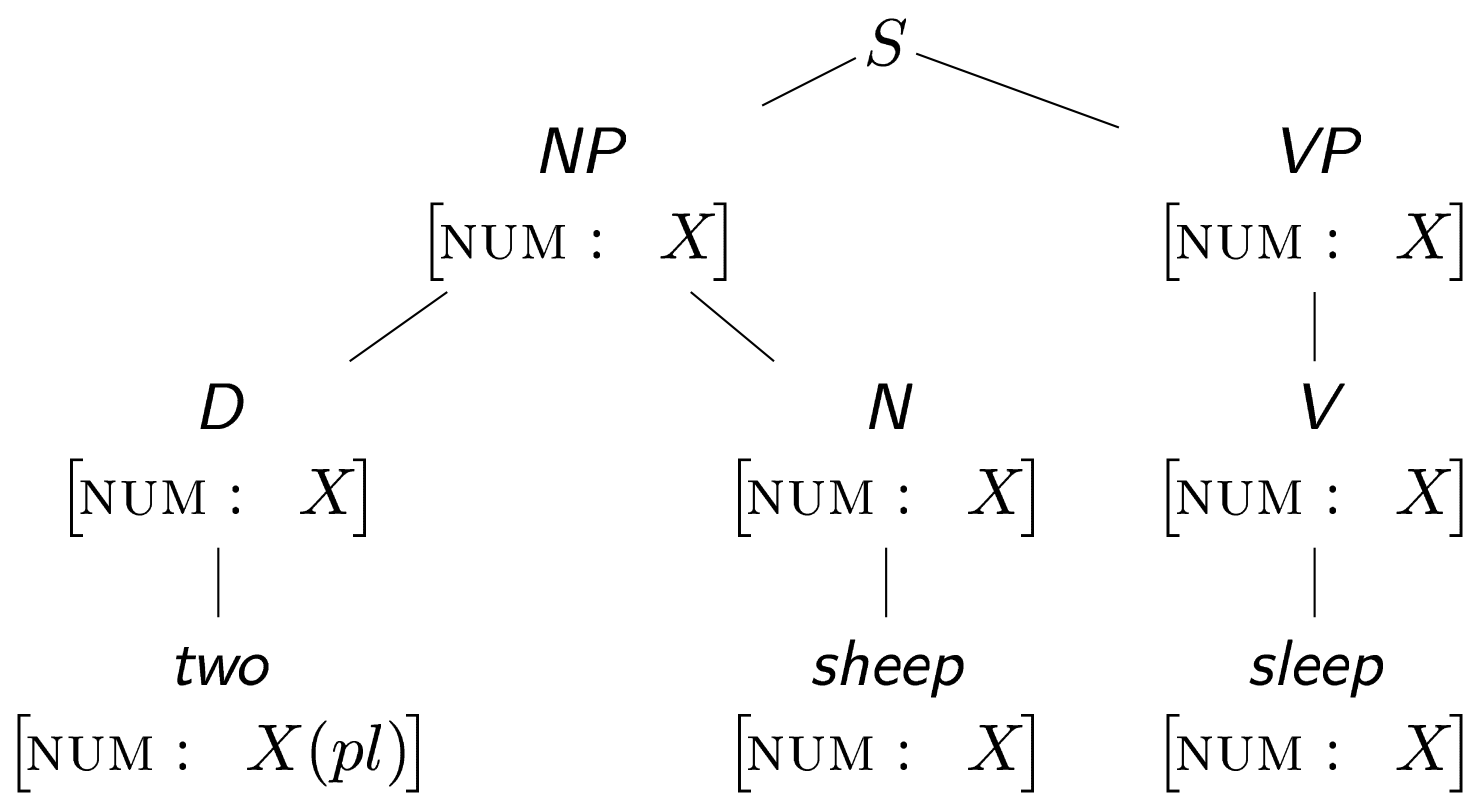
Derivation tree



Derivation tree



Derivation tree



Unification grammars: language

To determine whether a sequence of words, $w = a_1 \cdots a_n$, is in $L(G)$, let σ be some multi-AVM obtained by concatenating $A_1, \ldots, A_n$, where each A_i is a lexical entry of the word a_1.

Let σ' be a multi-AVM that is unifiable with σ: $\sigma \sqcup \sigma'$ does not fail.

$w \in L(G)$ if and only if there is a derivation in G whose first form consists of the start symbol, and whose last form is σ'.

A context-free grammar G_1'

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

$S_{sg} \rightarrow NP_{sg}\ VP_{sg}$	$S_{pl} \rightarrow NP_{pl}\ VP_{pl}$
$NP_{sg} \rightarrow D_{sg}\ N_{sg}$	$NP_{pl} \rightarrow D_{pl}\ N_{pl}$
$VP_{sg} \rightarrow V_{sg}$	$VP_{pl} \rightarrow V_{pl}$
$VP_{sg} \rightarrow V_{sg}\ NP_{sg} \mid V_{sg}\ NP_{pl}$	$VP_{pl} \rightarrow V_{pl}\ NP_{sg} \mid V_{pl}\ NP_{pl}$
$D_{sg} \rightarrow$ *a*	$D_{pl} \rightarrow$ *two*
$N_{sg} \rightarrow$ *lamb* | *sheep*	$N_{pl} \rightarrow$ *lambs* | *sheep*
$V_{sg} \rightarrow$ *sleeps*	$V_{pl} \rightarrow$ *sleep*

Imposing case control

$$\underset{\left[\begin{array}{ll}\text{NUM}: & X\\ \text{CASE}: & Y\end{array}\right]}{\textit{PropN}} \rightarrow \underset{\left[\begin{array}{ll}\text{NUM}: & X(sg)\\ \text{CASE}: & Y\end{array}\right]}{\textit{Rachel}} \;\Big|\; \underset{\left[\begin{array}{ll}\text{NUM}: & X(sg)\\ \text{CASE}: & Y\end{array}\right]}{\textit{Jacob}}$$

$$\underset{\left[\begin{array}{ll}\text{NUM}: & X\\ \text{CASE}: & Y\end{array}\right]}{\textit{Pron}} \rightarrow \underset{\left[\begin{array}{ll}\text{NUM}: & X(sg)\\ \text{CASE}: & Y(nom)\end{array}\right]}{\textit{she}} \;\Big|\; \underset{\left[\begin{array}{ll}\text{NUM}: & X(sg)\\ \text{CASE}: & Y(acc)\end{array}\right]}{\textit{her}}$$

Imposing case control

$$\underset{\begin{bmatrix}\mathrm{NUM}: & X\\ \mathrm{CASE}: & Y\end{bmatrix}}{\mathit{NP}} \rightarrow \underset{\begin{bmatrix}\mathrm{NUM}: & X\end{bmatrix}}{\mathit{D}} \quad \underset{\begin{bmatrix}\mathrm{NUM}: & X\\ \mathrm{CASE}: & Y\end{bmatrix}}{\mathit{N}}$$

$$\underset{\begin{bmatrix}\mathrm{NUM}: & X\\ \mathrm{CASE}: & Y\end{bmatrix}}{\mathit{NP}} \rightarrow \underset{\begin{bmatrix}\mathrm{NUM}: & X\\ \mathrm{CASE}: & Y\end{bmatrix}}{\mathit{PropN}}$$

$$\underset{\begin{bmatrix}\mathrm{NUM}: & X\\ \mathrm{CASE}: & Y\end{bmatrix}}{\mathit{NP}} \rightarrow \underset{\begin{bmatrix}\mathrm{NUM}: & X\\ \mathrm{CASE}: & Y\end{bmatrix}}{\mathit{Pron}}$$

Imposing case control

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

$$\underset{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}}{VP} \rightarrow \underset{\begin{bmatrix}\text{NUM}: & X\end{bmatrix}}{V} \quad \underset{\begin{bmatrix}\text{NUM}: & Y \\ \text{CASE}: & acc\end{bmatrix}}{NP}$$

Imposing subcategorization constraints

$$\underset{\left[\begin{array}{ll}\text{NUM}: & X \\ \text{SUBCAT}: & intrans\end{array}\right]}{V} \rightarrow \underset{\left[\text{NUM}: \; X(sg)\right]}{\textit{sleeps}} \;\Big|\; \underset{\left[\text{NUM}: \; X(pl)\right]}{\textit{sleep}}$$

$$\underset{\left[\begin{array}{ll}\text{NUM}: & X \\ \text{SUBCAT}: & trans\end{array}\right]}{V} \rightarrow \underset{\left[\text{NUM}: \; X(sg)\right]}{\textit{feeds}} \;\Big|\; \underset{\left[\text{NUM}: \; X(pl)\right]}{\textit{feed}}$$

Imposing subcategorization constraints

$$\underset{\left[\text{NUM}: \; X\right]}{\mathit{VP}} \rightarrow \underset{\begin{bmatrix}\text{NUM}: & X \\ \text{SUBCAT}: & intrans\end{bmatrix}}{\mathit{V}}$$

$$\underset{\left[\text{NUM}: \; X\right]}{\mathit{VP}} \rightarrow \underset{\begin{bmatrix}\text{NUM}: & X \\ \text{SUBCAT}: & trans\end{bmatrix}}{\mathit{V}} \quad \underset{\left[\text{NUM}: \; Y\right]}{\mathit{NP}}$$

Imposing subcategorization constraints

$$S$$

$$\stackrel{(1)}{\Rightarrow} \underset{\left[\text{NUM}: X\right]}{\textit{NP}} \quad \underset{\left[\text{NUM}: X\right]}{\textit{VP}}$$

$$\stackrel{(2)}{\Rightarrow} \underset{\left[\text{NUM}: X\right]}{\textit{D}} \quad \underset{\left[\text{NUM}: X\right]}{\textit{N}} \quad \underset{\left[\text{NUM}: X\right]}{\textit{VP}}$$

$$\stackrel{(4.2)}{\Rightarrow} \underset{\left[\text{NUM}: X\right]}{\textit{D}} \quad \underset{\left[\text{NUM}: X\right]}{\textit{N}} \quad \underset{\begin{bmatrix}\text{NUM}: & X\\ \text{SUBCAT}: & trans\end{bmatrix}}{\textit{V}} \quad \underset{\left[\text{NUM}: Y\right]}{\textit{NP}}$$

$$\stackrel{(1)}{\Rightarrow} \underset{\left[\text{NUM}: X\right]}{\textit{D}} \quad \underset{\left[\text{NUM}: X\right]}{\textit{N}} \quad \underset{\begin{bmatrix}\text{NUM}: & X\\ \text{SUBCAT}: & trans\end{bmatrix}}{\textit{V}} \quad \underset{\left[\text{NUM}: Y\right]}{\textit{D}} \quad \underset{\left[\text{NUM}: Y\right]}{\textit{N}}$$

$$\stackrel{*}{\Rightarrow} \underset{\left[\text{NUM}: sg\right]}{\textit{a}} \quad \underset{\left[\text{NUM}: sg\right]}{\textit{shepherd}} \quad \underset{\begin{bmatrix}\text{NUM}: & sg\\ \text{SUBCAT}: & trans\end{bmatrix}}{\textit{feeds}} \quad \underset{\left[\text{NUM}: pl\right]}{\textit{two}} \quad \underset{\left[\text{NUM}: pl\right]}{\textit{sheep}}$$

G_2, a complete E_0-grammar

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

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

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

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

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

G_2, a complete E_0-grammar

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

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

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

G_2, a complete E_0-grammar

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

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

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

Internalizing categories

The rule

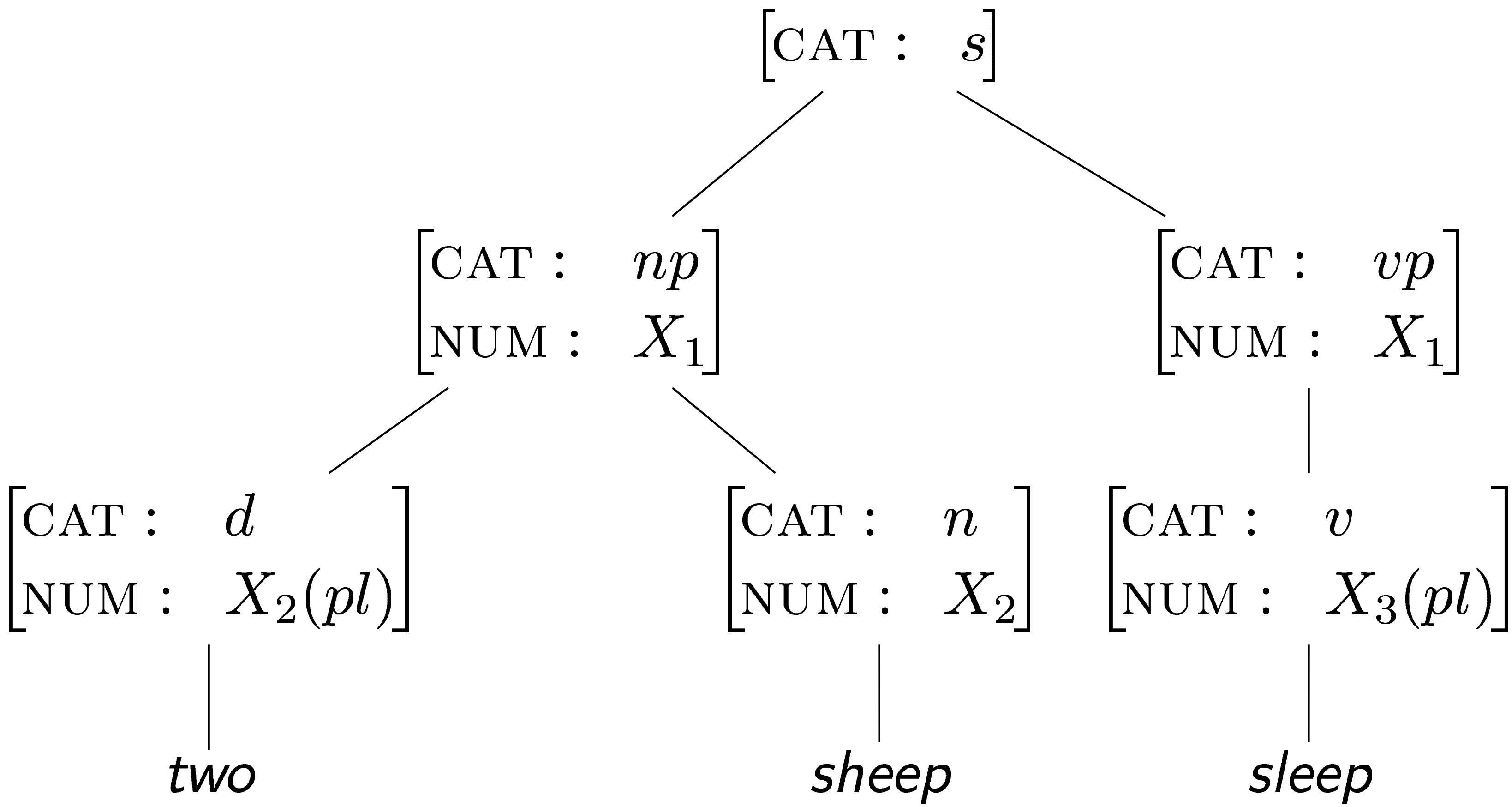
$$\underset{\left[\textsc{num}: \quad X\right]}{\mathit{NP}} \rightarrow \underset{\left[\textsc{num}: \quad X\right]}{\mathit{D}} \quad \underset{\left[\textsc{num}: \quad X\right]}{\mathit{N}}$$

can be re-written as

$$\begin{bmatrix}\textsc{cat}: & np\\ \textsc{num}: & X\end{bmatrix} \rightarrow \begin{bmatrix}\textsc{cat}: & d\\ \textsc{num}: & X\end{bmatrix} \begin{bmatrix}\textsc{cat}: & n\\ \textsc{num}: & X\end{bmatrix}$$

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

Internalizing categories



Internalizing categories

Base categories do not have to be atomic any more. This facilitates generalizations over categories:

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

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

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

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

Internalizing categories

Finitely ambiguous backbone (or skeleton)

Additional expressive power:

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

Internalizing categories thus results in a powerful mechanism for specifying linguistic generalizations.

Subcategorization lists

A more sophisticated solution using internalized categories:

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

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

Subcategorization lists

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

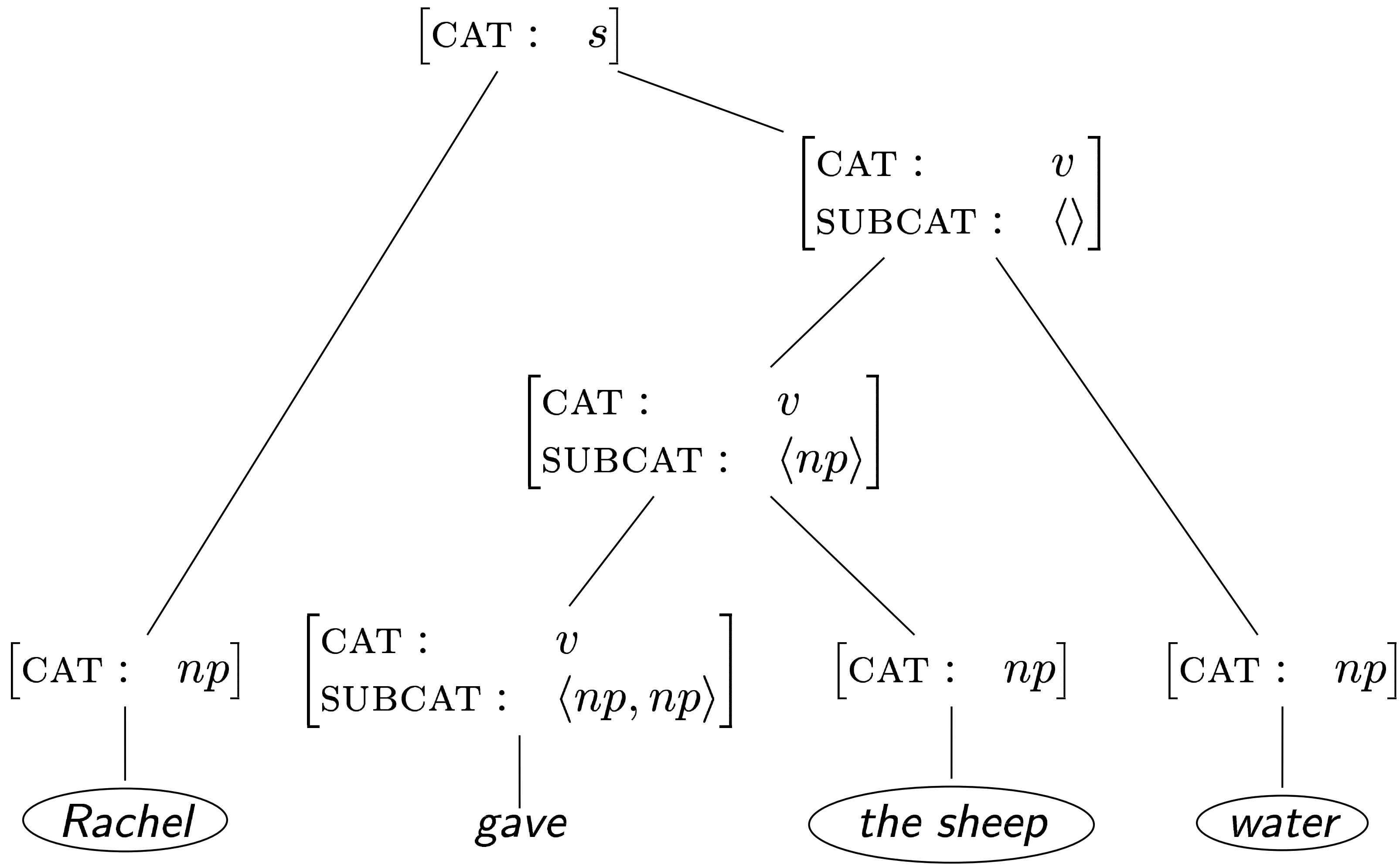
$$\textit{tell} \quad \begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \left\langle \begin{bmatrix}\text{CAT}: & np\end{bmatrix}, \begin{bmatrix}\text{CAT}: & s\end{bmatrix} \right\rangle \\ \text{NUM}: & pl \end{bmatrix}$$

Subcategorization lists

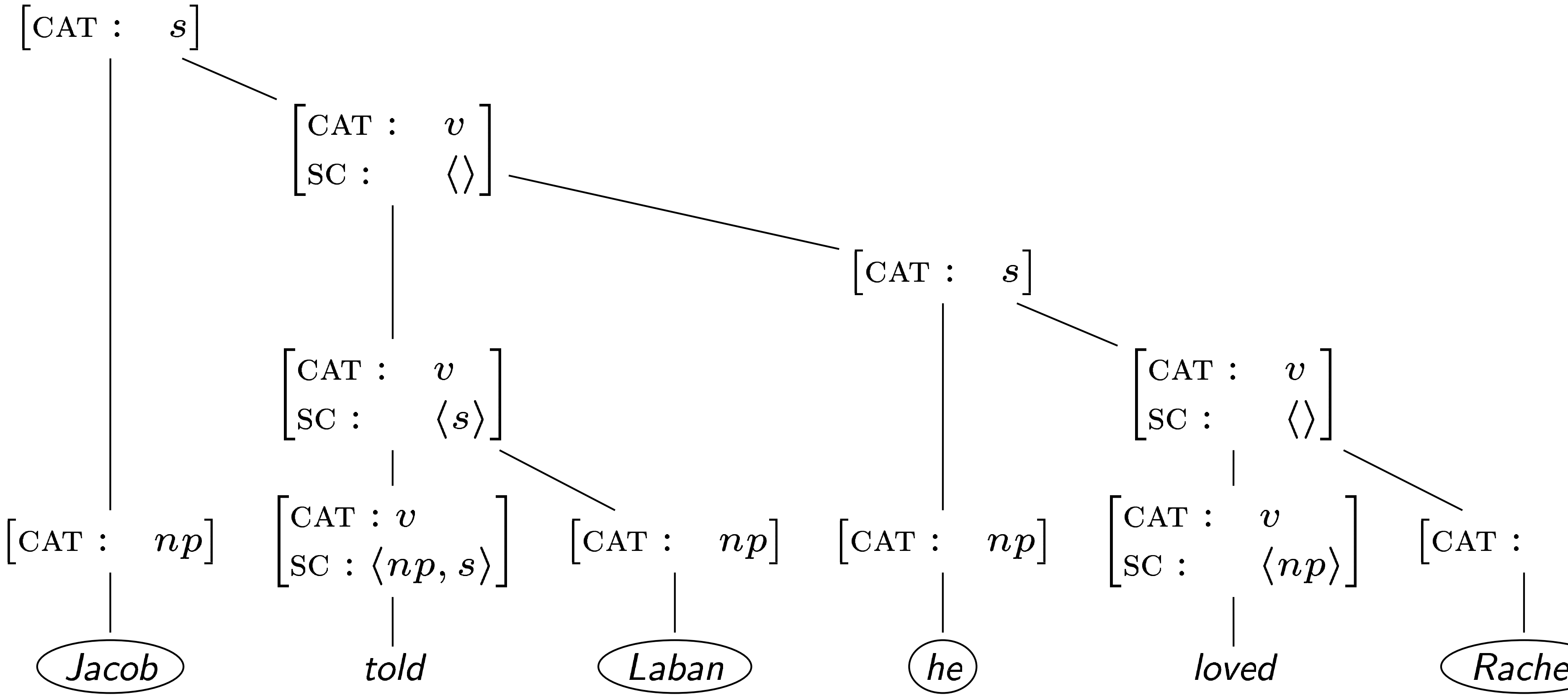
$$\begin{bmatrix}\text{CAT}: & s\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & np\end{bmatrix} \begin{bmatrix}\text{CAT}: & v \\ \text{SUBCAT}: & elist\end{bmatrix}$$

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

Subcategorization lists



Subcategorization lists



Subcategorization lists

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

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

G_3, a complete E_1-grammar

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

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

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

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

G_3, a complete E_1-grammar

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

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

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

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

G_3, a complete E_1-grammar

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

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

$$\textit{she} \rightarrow \begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & sg \\ \text{CASE}: & nom \end{bmatrix}$$

$$\textit{her} \rightarrow \begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & pl \\ \text{CASE}: & acc \end{bmatrix}$$

G_3, a complete E_1-grammar

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

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

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

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

Long distance dependencies

The problem:

The shepherd wondered whom Jacob loved ␣.

An extension of G_2 that can handle such phenomena

Signaling that a constituent, in this case a noun phrase, is missing: *slash* features

Long distance dependencies

Start with the following rules of G_2:

$$(1) \quad \begin{bmatrix} \text{CAT}: & s \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & X \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & vp \\ \text{NUM}: & X \end{bmatrix}$$

$$(2) \quad \begin{bmatrix} \text{CAT}: & vp \\ \text{NUM}: & X \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & X \\ \text{SUBCAT}: & trans \end{bmatrix} \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & Y \\ \text{CASE}: & acc \end{bmatrix}$$

Long distance dependencies

To these two rules we add the following two:

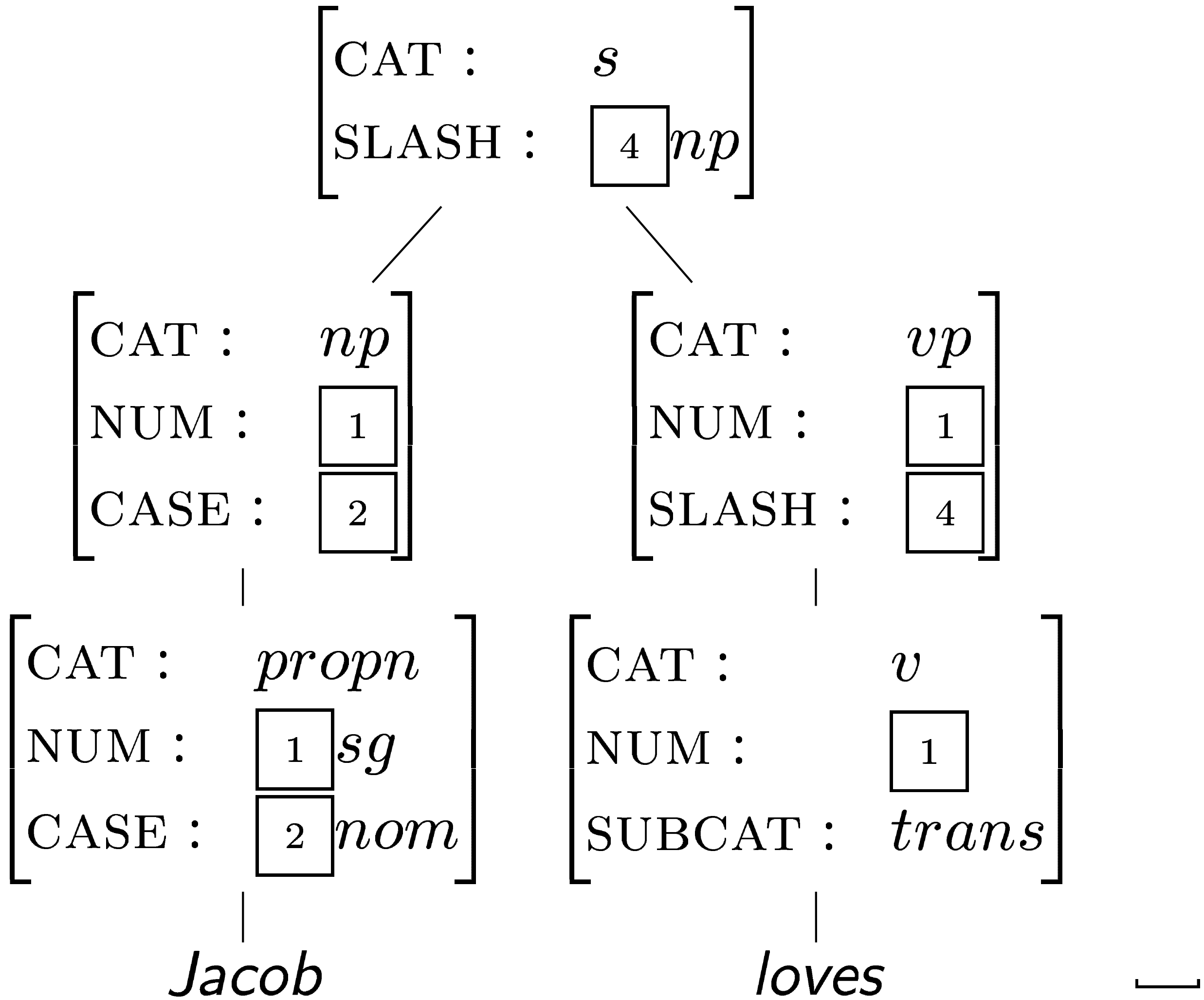
$$(3) \quad \begin{bmatrix} \text{CAT}: & s \\ \text{SLASH}: & Z \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & X \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & vp \\ \text{NUM}: & X \\ \text{SLASH}: & Z \end{bmatrix}$$

$$(4) \quad \begin{bmatrix} \text{CAT}: & vp \\ \text{NUM}: & X \\ \text{SLASH}: & np \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & X \\ \text{SUBCAT}: & trans \end{bmatrix}$$

Long distance dependencies

With the two additional rules, it is possible to derive partial phrases such as *Jacob loves* ␣:

Long distance dependencies



Long distance dependencies

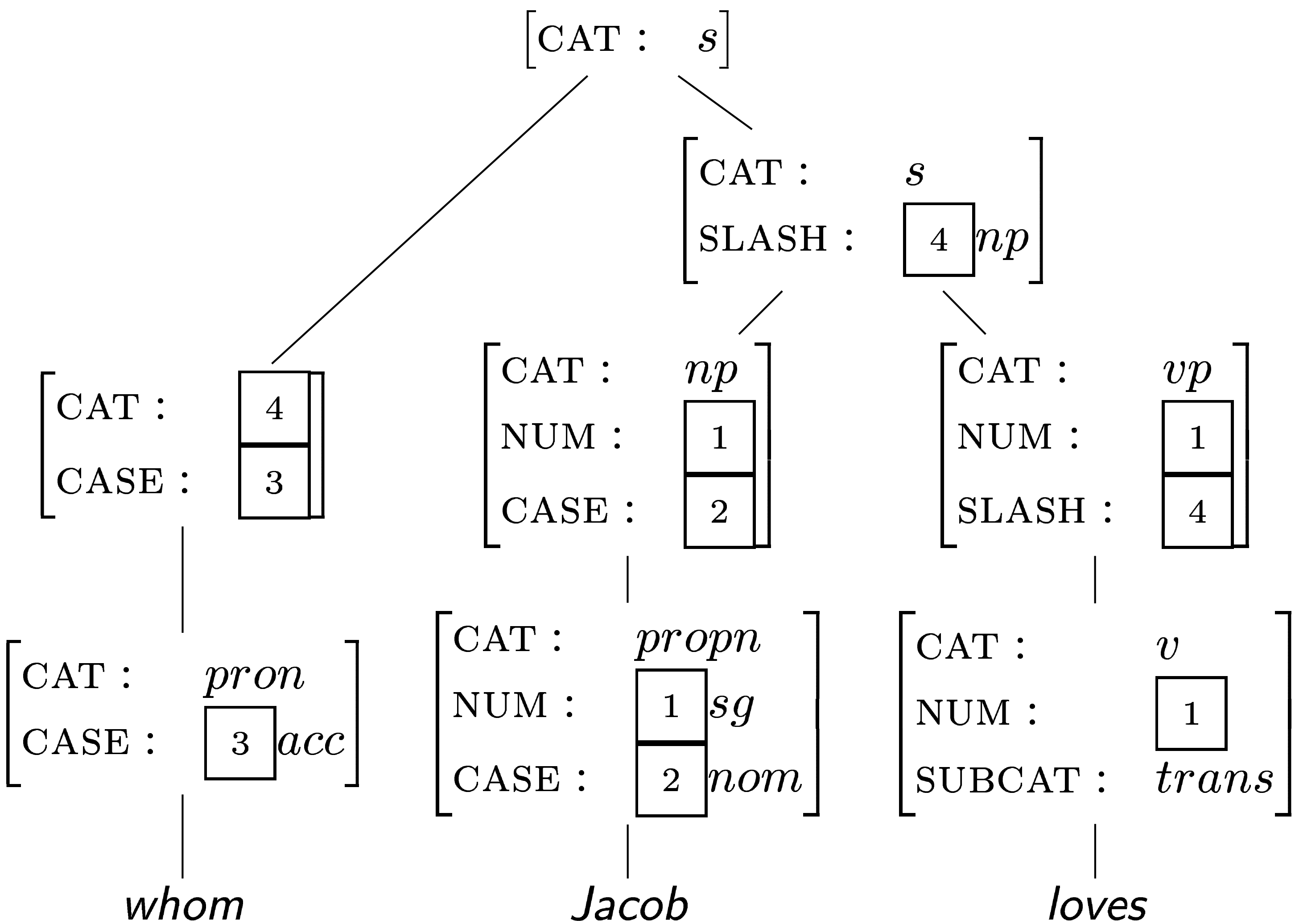
A rule to create "complete" sentences by combining the missing category with a "slashed" sentence:

$$(5) \quad \begin{bmatrix}\textsc{cat}: & s\end{bmatrix} \rightarrow \begin{bmatrix}\textsc{cat}: & Z\end{bmatrix} \begin{bmatrix}\textsc{cat}: & s \\ \textsc{slash}: & Z\end{bmatrix}$$

Long distance dependencies

A derivation tree for *whom Jacob loves* ␣:

Long distance dependencies



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 ␣.

Long distance dependencies

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

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:

$$(6) \quad \begin{bmatrix} \text{CAT}: & vp \\ \text{SLASH}: & Z \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & s \end{bmatrix} \begin{bmatrix} \text{CAT}: & s \\ \text{SLASH}: & Z \end{bmatrix}$$

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

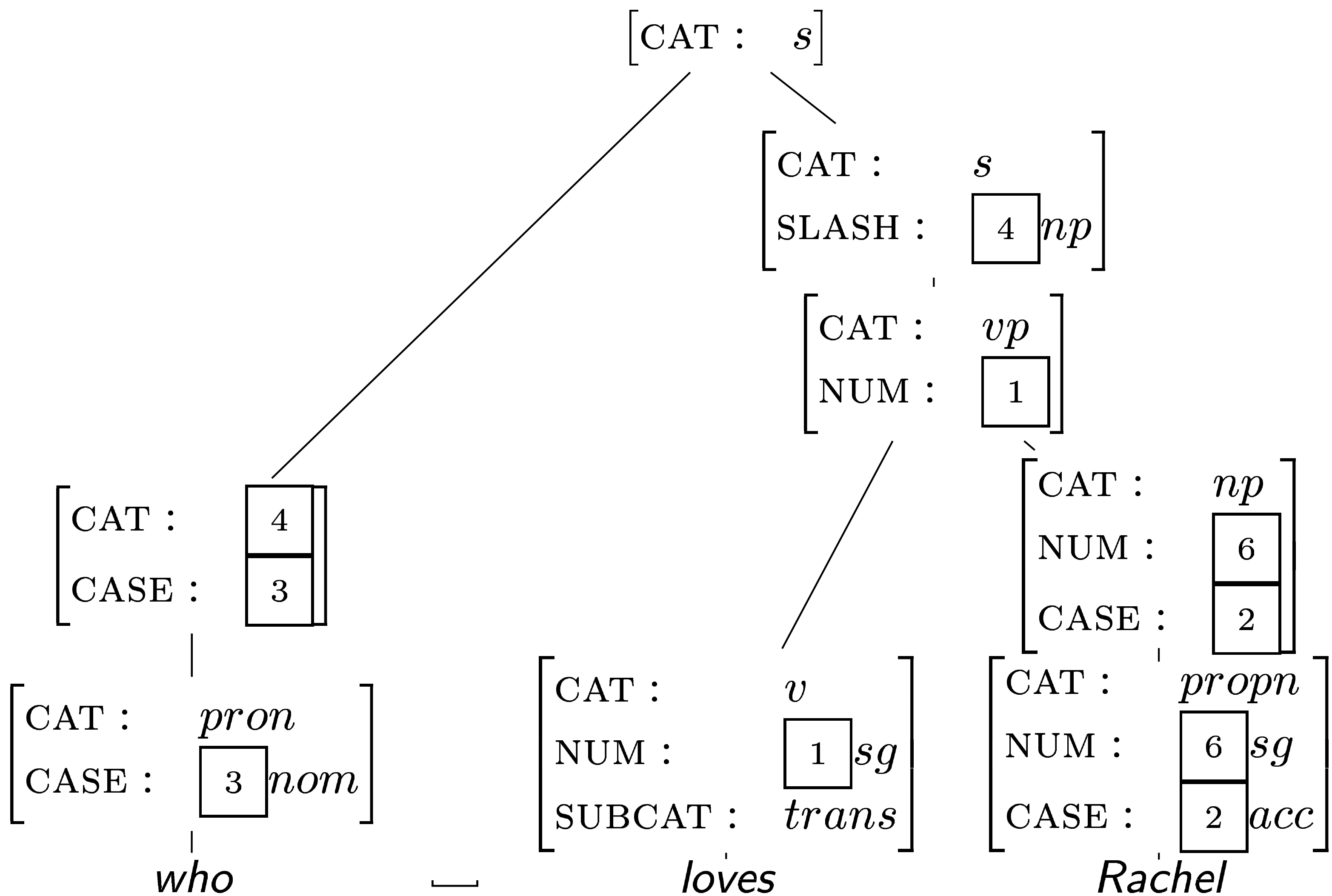
Long distance dependencies

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

$$(8) \quad \begin{bmatrix} \text{CAT}: & s \\ \text{SLASH}: & np \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & vp \\ \text{NUM}: & X \end{bmatrix}$$

A derivation tree for *who* ␣ *loves Rachel*:

Long distance dependencies



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 the flock of Laban his mother's brother].*

Constituent coordination

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

Assumptions:

- Every category of E_0 can be conjoined
- The same conjunctions (and, or, but) can be used for all the categories

Constituent coordination

A CFG solution:

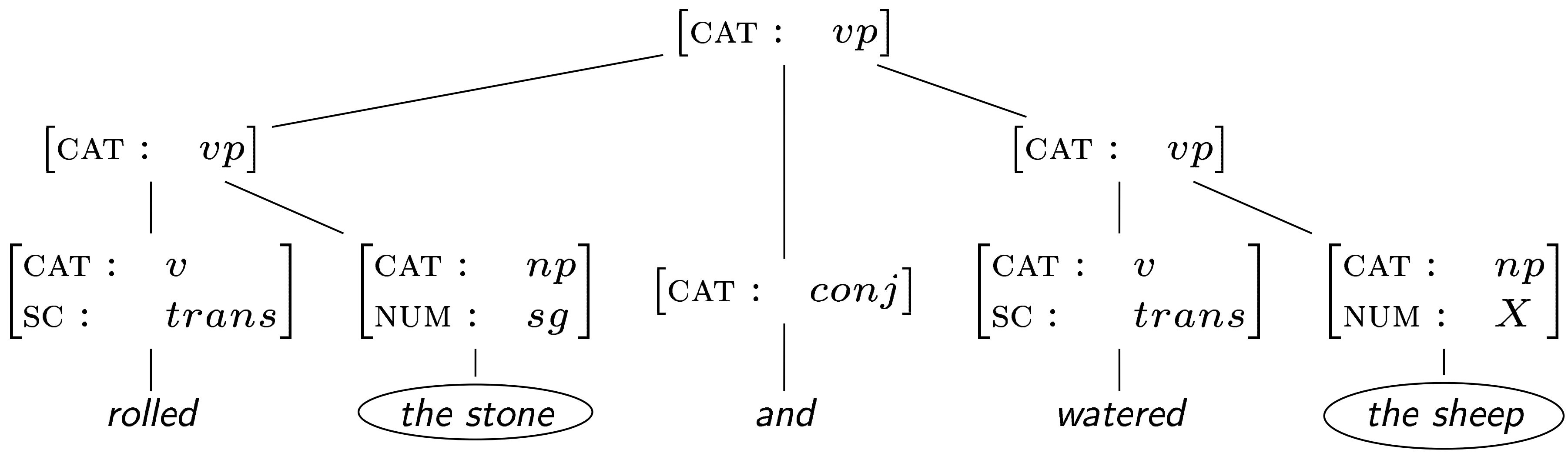
$$\begin{array}{ccl} \text{S} & \rightarrow & \text{S Conj S} \\ \text{NP} & \rightarrow & \text{NP Conj NP} \\ \text{VP} & \rightarrow & \text{VP Conj VP} \\ \vdots & & \\ \text{Conj} & \rightarrow & \textit{and},\ \textit{or},\ \textit{but},\ \ldots \end{array}$$

Constituent coordination

With generalized categories:

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

Constituent coordination

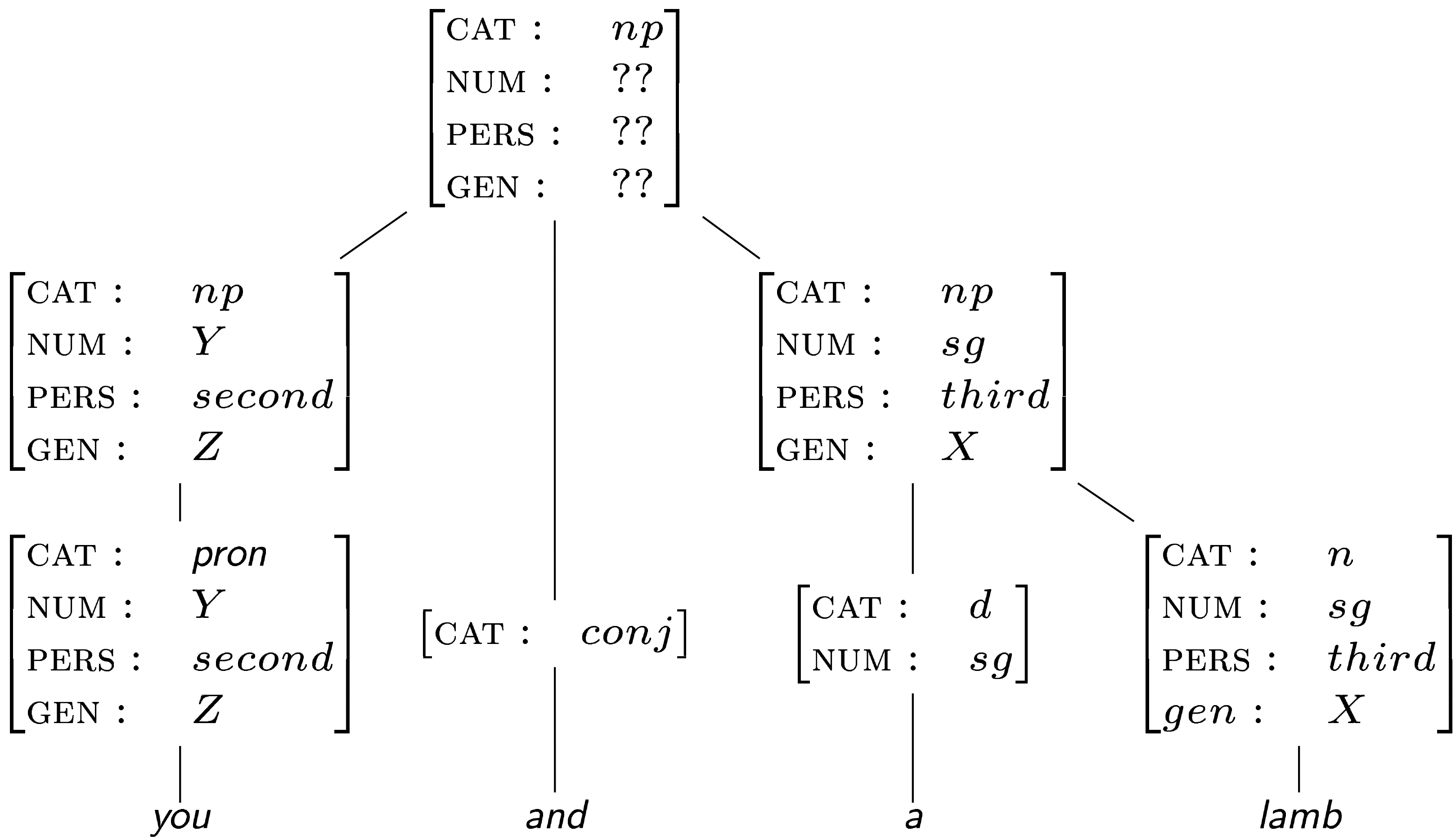


Coordination

Problems:

- Properties of conjoined constituents
- Non-constituent coordination

Properties of conjoined constituents



Non-constituent coordination

Joseph became wealthy

Joseph became a minister

Joseph became [wealthy and a minister]

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

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

Rachel [kissed] and Jacob [hugged] Binyamin

The expressive power of unification grammars

Unification grammars are strictly more expressive than context-free grammars

A unification grammar for $L = \{a^n b^n c^n \mid n > 0\}$: count the number of ‘a’s, ‘b’s or ‘c’s in a unary base

The string bbb is derived by the following AVM:

$$\begin{bmatrix} cat: & b \\ t: & \begin{bmatrix} t: & \begin{bmatrix} t: & 1 \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The expressive power of unification grammars

$$\begin{bmatrix} cat: & s \end{bmatrix} \rightarrow \begin{bmatrix} cat: & a \\ t: & X \end{bmatrix} \begin{bmatrix} cat: & b \\ t: & X \end{bmatrix} \begin{bmatrix} cat: & c \\ t: & X \end{bmatrix}$$

$$\begin{bmatrix} cat: & a \\ t: & \begin{bmatrix} t: & X \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} cat: & at \end{bmatrix} \begin{bmatrix} cat: & a \\ t: & X \end{bmatrix}$$

$$\begin{bmatrix} cat: & a \\ t: & 1 \end{bmatrix} \rightarrow \begin{bmatrix} cat: & at \end{bmatrix}$$

$$\begin{bmatrix} cat: & b \\ t: & \begin{bmatrix} t: & X \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} cat: & bt \end{bmatrix} \begin{bmatrix} cat: & b \\ t: & X \end{bmatrix}$$

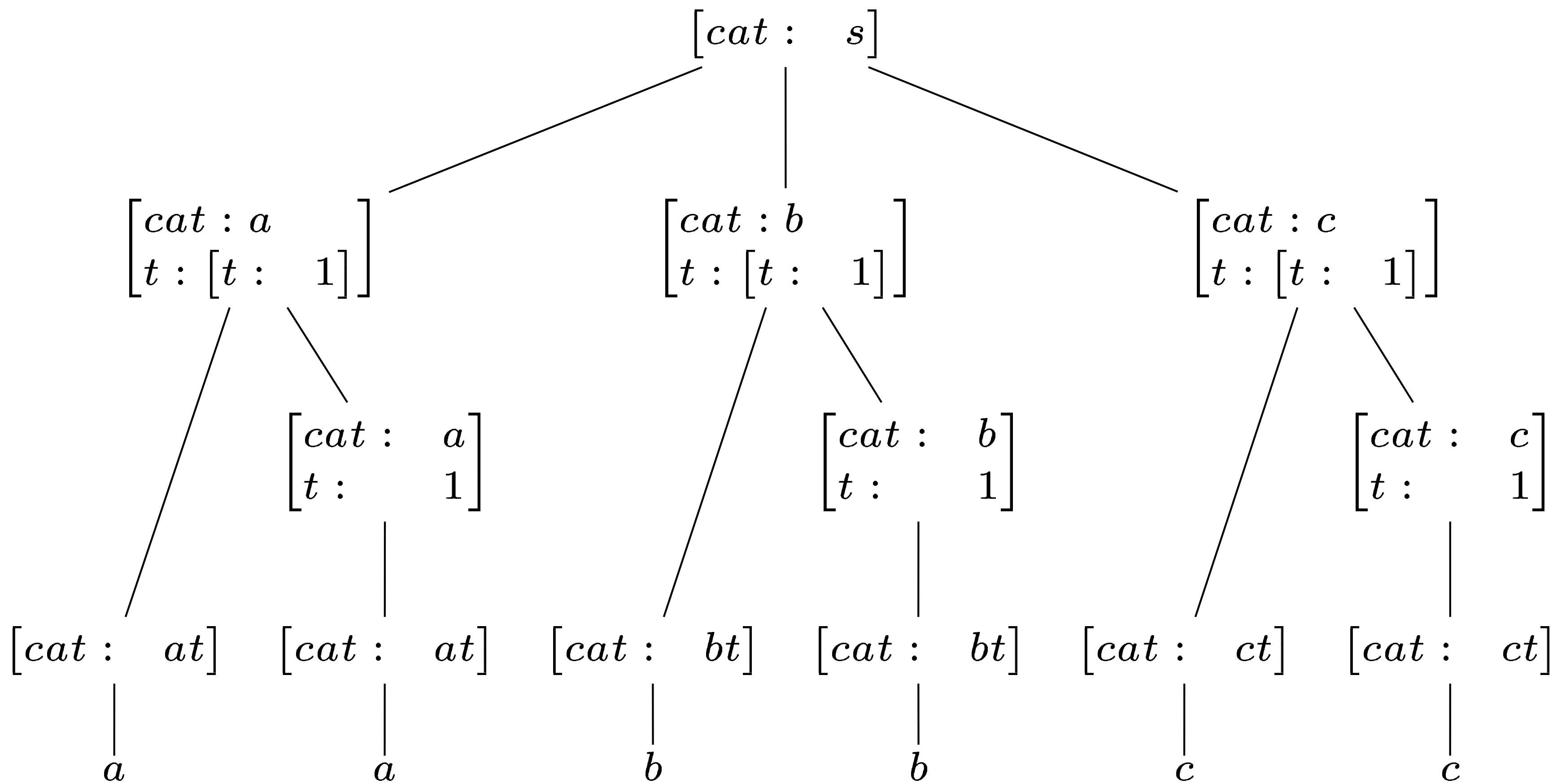
$$\begin{bmatrix} cat: & b \\ t: & 1 \end{bmatrix} \rightarrow \begin{bmatrix} cat: & bt \end{bmatrix}$$

$$\begin{bmatrix} cat: & c \\ t: & \begin{bmatrix} t: & X \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} cat: & ct \end{bmatrix} \begin{bmatrix} cat: & c \\ t: & X \end{bmatrix}$$

$$\begin{bmatrix} cat: & c \\ t: & 1 \end{bmatrix} \rightarrow \begin{bmatrix} cat: & ct \end{bmatrix}$$

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

The expressive power of unification grammars



The expressive power of unification grammars

Unification grammars are equivalent in their weak generative power to *unrestricted rewriting systems*.

This is equivalent to saying that unification grammars are equivalent to Turing machines in their generative capacity, or that the languages generated by unification grammars are exactly the set of *recursively enumerable* languages.

Given an arbitrary unification grammar G and a string w, no computational procedure can be designed to determine whether $w \in L(G)$.

The expressive power of unification grammars

A **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
- $\delta : (Q \setminus \{h\}) \times \Sigma \to Q \times (\Sigma \cup \{L, R\})$ is a total function specifying transitions.

The expressive power of unification grammars

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.

The expressive power of unification grammars

Let

$$\textit{first}(\sigma_1 \cdots \sigma_n) = \begin{cases} \sigma_1 & n > 0 \\ \flat & n = 0 \end{cases}$$

$$\textit{but-first}(\sigma_1 \cdots \sigma_n) = \begin{cases} \sigma_2 \cdots \sigma_n & n > 0 \\ \epsilon & n = 0 \end{cases}$$

$$\textit{last}(\sigma_1 \cdots \sigma_n) = \begin{cases} \sigma_n & n > 0 \\ \flat & n = 0 \end{cases}$$

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

The expressive power of unification grammars

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

$$\begin{array}{ll}
(p, w_l, \sigma', w_r) & \text{if } \delta(q,\sigma) = (p, \sigma'),\ \sigma' \in \Sigma \\
(p, w_l\sigma, \textit{first}(w_r), \textit{but-first}(w_r)) & \text{if } \delta(q,\sigma) = (p, R) \\
(p, \textit{but-last}(w_l), \textit{last}(w_l), \sigma w_r) & \text{if } \delta(q,\sigma) = (p, L)
\end{array}$$

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.

The expressive power of unification grammars

A grammar can "simulate" the operation of a Turing machine.

Define a unification grammar G_M for every Turing machine M such that:

$$L(G_M) = \begin{cases} \{\text{halt}\} & \text{if } M \text{ terminates on the empty input} \\ \emptyset & \text{otherwise} \end{cases}$$

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, thus determining whether M terminates for the empty input, which is known to be undecidable.

The expressive power of unification grammars

Let $M = (Q, \Sigma, \flat, \delta, s, h)$ be a Turing machine.

Define a unification grammar G_M as follows:

- $\textsc{Feats} = \{$*left, right, curr, first, rest*$\}$
- $\textsc{Atoms} = \Sigma \cup \{$*elist*$\}$
- The base categories of the grammar are the states Q, with an additional symbol $S \notin Q$
- There is only one terminal symbol, halt

The expressive power of unification grammars

The grammar rules can be divided to four groups.

First, two rules are defined for every Turing machine:

$$\begin{aligned} \mathsf{S} \quad &\rightarrow \quad \overset{\mathsf{s}}{\begin{bmatrix} curr: & \flat \\ right: & elist \\ left: & elist \end{bmatrix}} \\ h \quad &\rightarrow \quad \text{halt} \end{aligned}$$

The expressive power of unification grammars

The second group of rules are defined for rewriting transitions. For every q, σ such that $\delta(q, \sigma) = (p, \sigma')$ and $\sigma' \in \Sigma$, the following rule is defined:

$$\overset{q}{\begin{bmatrix} curr: & \sigma \\ right: & X \\ left: & Y \end{bmatrix}} \rightarrow \overset{p}{\begin{bmatrix} curr: & \sigma' \\ right: & X \\ left: & Y \end{bmatrix}}$$

The expressive power of unification grammars

A third group of rules is defined for right movement of the head. For every q, σ such that $\delta(q,\sigma) = (p, R)$ define two rules:

$$\overset{q}{\begin{bmatrix} curr: & \sigma \\ right: & elist \\ left: & X \end{bmatrix}} \rightarrow \overset{p}{\begin{bmatrix} curr: & \flat \\ right: & elist \\ left: & \begin{bmatrix} first: & \sigma \\ rest: & X \end{bmatrix} \end{bmatrix}}$$

$$\overset{q}{\begin{bmatrix} curr: & \sigma \\ right: & \begin{bmatrix} first: & X \\ rest: & Y \end{bmatrix} \\ left: & W \end{bmatrix}} \rightarrow \overset{p}{\begin{bmatrix} curr: & X \\ right: & Y \\ left: & \begin{bmatrix} first: & \sigma \\ rest: & W \end{bmatrix} \end{bmatrix}}$$

The expressive power of unification grammars

The last group of rules handle left movements in a symmetric fashion. For every q, σ such that $\delta(q,\sigma) = (p, L)$ define two rules:

$$\overset{q}{\begin{bmatrix} curr: & \sigma \\ right: & X \\ left: & elist \end{bmatrix}} \rightarrow \overset{p}{\begin{bmatrix} curr: & \flat \\ right: & \begin{bmatrix} first: & \sigma \\ rest: & X \end{bmatrix} \\ left: & elist \end{bmatrix}}$$

$$\overset{q}{\begin{bmatrix} curr: & \sigma \\ right: & X \\ left: & \begin{bmatrix} first: & Y \\ rest: & W \end{bmatrix} \end{bmatrix}} \rightarrow \overset{p}{\begin{bmatrix} curr: & Y \\ right: & \begin{bmatrix} first: & \sigma \\ rest: & X \end{bmatrix} \\ left: & W \end{bmatrix}}$$

The expressive power of unification grammars

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

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

The universal recognition problem for unification grammars is undecidable.