Context-free grammars for natural languages

- Context-free grammars can be used for a variety of syntactic constructions, including some non-trivial phenomena such as unbounded dependencies, extraction, extraposition etc.
- However, some (formal) languages are not context-free, and therefore there are certain sets of strings that cannot be generated by context-free grammars.
- The interesting question, of course, involves natural languages: are there natural languages that are not context-free? Are context-free grammars sufficient for generating every natural language?

A context-free grammar, G_0, for E_0

Example

A context-free grammar, G_0, for E_0

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

Context-free grammars for natural languages

There are two major problems with this grammar.

1. it ignores the valence of verbs: there is no distinction among subcategories of verbs, and an intransitive verb such as *sleep* might occur with a noun phrase complement, while a transitive verb such as *love* might occur without one. In such a case we say that the grammar *overgenerates*: it generates strings that are not in the intended language.
2. there is no treatment of subject–verb agreement, so that a singular subject such as *the cat* might be followed by a plural form of verb such as *smile*. This is another case of overgeneration.

Both problems are easy to solve.

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 (subcategorization constraints are not imposed):

the lambs sleep

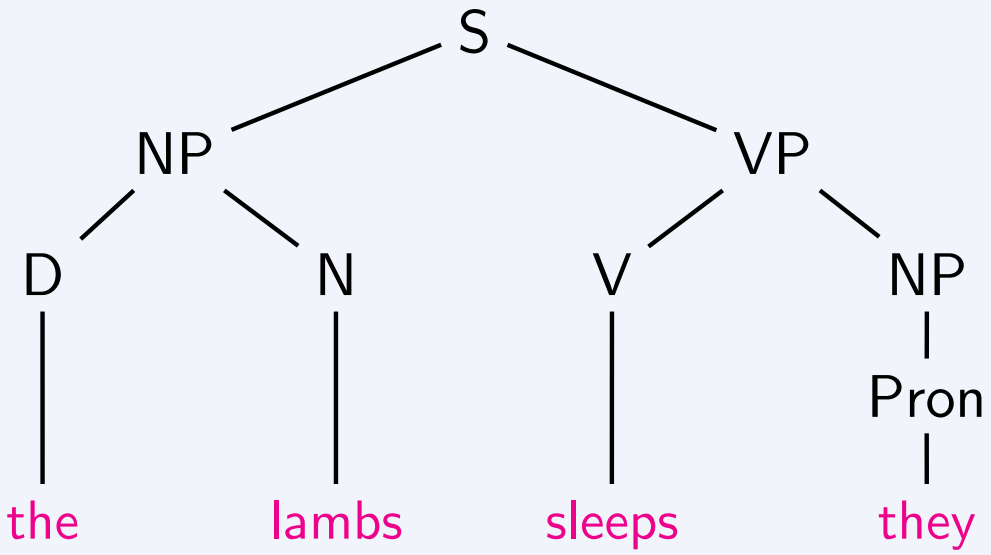
Jacob loves Rachel

$*$*the lambs sleep the sheep*

$*$*Jacob loves*

Problems of G_0

Example (Over-generation)



Verb valence

- To account for valence, we can replace the non-terminal symbol V by a set of symbols: *Vtrans*, *Vintrans*, *Vditrans* etc.
- We must also change the grammar rules accordingly:

Example

$VP \rightarrow Vintrans$	$Vintrnas \rightarrow$ *sleep, sleeps*
$VP \rightarrow Vtrans\ NP$	$Vtrans \rightarrow$ *love, loves*
$VP \rightarrow Vditrans\ NP\ NP$	$Vditrans \rightarrow$ *give, gives*

Agreement

- To account for agreement, we can again extend the set of non-terminal symbols such that categories that must agree reflect in the non-terminal that is assigned for them the features on which they agree.
- In the very simple case of English, it is sufficient to multiply the set of "nominal" and "verbal" categories, so that we get *Dsg, Dpl, Nsg, Npl, NPsg, NPpl, Vsg, Vlp, VPsg, VPpl* etc. We must also change the set of rules accordingly:

Agreement

Example

$Nsg \rightarrow$ *lamb*	$Npl \rightarrow$ *lambs*
$Nsg \rightarrow$ *sheep*	$Npl \rightarrow$ *sheep*
$Vsg \rightarrow$ *sleeps*	$Vpl \rightarrow$ *sleep*
$Vsg \rightarrow$ *smiles*	$Vpl \rightarrow$ *smile*
$Vsg \rightarrow$ *loves*	$Vpl \rightarrow$ *love*
$Vsg \rightarrow$ *saw*	$Vpl \rightarrow$ *saw*
$Dsg \rightarrow$ *a*	$Dpl \rightarrow$ *two*

Agreement

Example

$$S \rightarrow NPsg\ VPsg \qquad S \rightarrow NPpl\ VPpl$$
$$NPsg \rightarrow Dsg\ Nsg \qquad NPpl \rightarrow Dpl\ Npl$$
$$VPsg \rightarrow Vsg \qquad VPpl \rightarrow Vpl$$
$$VPsg \rightarrow VPsg\ NP \qquad VPpl \rightarrow VPpl\ NP$$

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.

Feature structures

- Motivated by the violations of the context-free grammar G_0, we would like to extend the CFG formalism with additional mechanisms that will facilitate the expression of information that is missing in G_0 in a uniform and compact way.
- 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*.

Overview

- An overview of feature structures, motivating their use as a representation of linguistic information
- Four different views of these entities:
  - *feature graphs*
  - *feature structures*
  - *attribute-value matrices* (AVMs)
- Feature structures in a broader context.

Motivation

- Words in natural languages have properties
- We want to model these properties in the lexicon
- We would like to associate with words not just atomic symbols, as in CFGs, but rather structural information that reflects their properties.

A simple lexicon

Example (A simple lexicon)

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

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

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

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

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

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

Feature structures

- Feature structures map *features* into *values*, which are themselves feature structures
- A special case of feature structures are *atoms*, which represent structureless values.
- For example, to deal with *number* (and impose its *agreement*), we use a feature NUM, and a set of atomic feature structures $\{sg,pl\}$ as its values, representing singularity and plurality, respectively.
- When a value is not atomic, it is *complex*.
- A complex value is, recursively, a feature structure consisting of features and values.

A complex feature structure

Example (A complex feature structure)

loves: $\begin{bmatrix} \text{VTYPE}: & \textit{transitive} \\ \text{AGR}: & \begin{bmatrix} \text{NUM}: & \textit{sg} \\ \text{PERS}: & \textit{third} \end{bmatrix} \end{bmatrix}$

Grouping features

- Deciding how to group features is up to the grammar designer, and is intended to capture syntactic generalizations.
- If NUMBER and PERSON 'go together' in formulating restrictions, it is more appropriate to group them as in this example.
- Moreover, such a grouping might be beneficial when feature structures are being modified.
- Processes of *derivation* and *parsing* (the application of grammar rules) are able to manipulate feature structures to reflect application of such constraints.
- When the properties of some feature structure are changed, it is possible to change the value of only *one* feature, namely AGR, rather than specify two separate changes for each subfeature.

Grouping features

- In the example lexicon, the lexical ambiguity of sheep is represented by an empty feature structure as the value of the NUM feature.
- This is interpreted as the value of this feature being unconstrained.
- However, it would have been useful to be able to state that the only possible values for this feature are, say, *sg* and *pl*.
- There are at least two different ways to specify such information:
  - by listing a set of values for the feature;
  - or by restricting its value to a certain "type" of permissible values.
- We do not explore the former solution here.
- The latter solution is employed by *typed feature structure formalisms*.

Adding features to phrases

- *Words* are not the only linguistic entities that have properties; words are combined into *phrases*, and those also have properties which can be modeled by feature:value pairs.
- For example, the noun phrase a sheep has the value *sg* for the NUM feature, while two sheep has the value *pl* for NUM.
- Consequently, grammar non-terminals, too, must be decorated with features, representing the endowment of phrases of this category with that feature.

Feature graphs

- The informal discussion of feature structures above depicted them using a representation, called *attribute-value matrices* (AVMs), which is common in the linguistic literature.
- We begin the discussion of feature structures by defining the concept of *feature graphs*, using well-known concepts of graph theory.
- A graph view of feature structures facilitates computational processing because so many properties of graphs are well understood and because graphs lend themselves to efficient processing.
- We will return to AVMs and discuss their correspondence with feature graphs later on.

Definitions

- Feature graphs are defined over a *signature* consisting of non-empty, finite, disjoint sets FEATS of features and ATOMS of atoms.
- Features are used to encode properties of (linguistic) objects, such as NUMBER, GENDER etc.
- Atoms are used for the (atomic) values of such features, as in *plural, feminine* etc.
- We use a convention of depicting features in SMALL CAPITALS and atoms in *italics*.

Signature

Definition (Signature)

A **signature** is a structure $\mathcal{S} = \langle \textsc{Atoms}, \textsc{Feats} \rangle$, where $\textsc{Atoms}$ is a finite set of atoms and $\textsc{Feats}$ is a finite set of features.

- We assume some fixed signature throughout this presentation.
- Meta-variables f, g (with or without subscripts or superscripts) range over features, and a, b, etc. over atoms.
- We usually assume that both $\textsc{Feats}$ and $\textsc{Atoms}$ are non-empty (and sometimes even assume that they include more than one element each).

Feature graphs

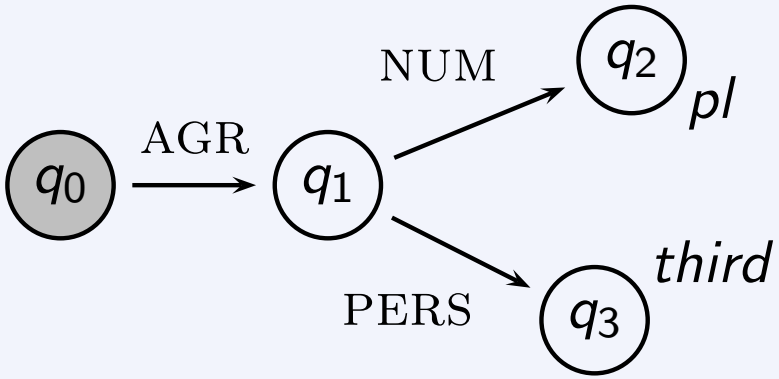
Definition (Feature graphs)

A **feature graph** $A = \langle Q_A, \bar{q}_A, \delta_A, \theta_A \rangle$ is a finite, directed, connected, labeled graph consisting of a finite, nonempty set of nodes Q_A (such that $Q_A \cap \textsc{Feats} = Q_A \cap \textsc{Atoms} = \emptyset$), a root $\bar{q}_A \in Q_A$, a partial function $\delta_A : Q_A \times \textsc{Feats} \rightarrow Q_A$ specifying the arcs such that every node $q \in Q_A$ is accessible from $\bar{q}_A$, and a partial function, marking some of the sinks: $\theta_A : Q_S \rightarrow \textsc{Atoms}$, where $Q_S = \{q \in Q_A \mid \delta_A(q, f)\uparrow \text{ for every } f\}$.
Given a signature of features $\textsc{Feats}$ and atoms $\textsc{Atoms}$, let $\mathcal{G}(\textsc{Feats}, \textsc{Atoms})$ be the set of all feature graphs over the signature.

Feature graphs

Example (Feature graphs)

The graph displayed below is $\langle Q, \bar{q}, \delta, \theta \rangle$, where $Q = \{q_0, q_1, q_2, q_3\}, \bar{q} = q_0, \delta(q_0, \text{AGR}) = q_1, \delta(q_1, \text{NUM}) = q_2, \delta(q_1, \text{PERS}) = q_3, Q_S = \{q_2, q_3\}, \theta(q_2) = pl, \theta(q_3) = third.$



Feature graphs

- The arcs of a feature graph are thus labeled by features.
- The root is a designated node from which all other nodes are accessible (through δ); note that nothing prevents the root from having incoming arcs.
- Sink nodes (nodes with no outgoing edges) can be marked by an atom, but can also be unmarked.

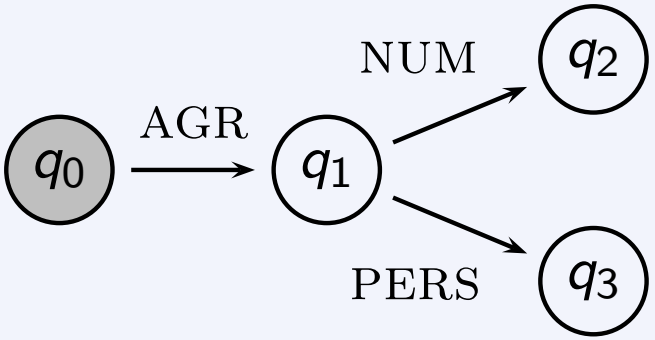
Feature graphs

- We use meta-variables A, B (with or without subscripts) to refer to feature graphs.
- We use $Q, \bar{q}, \delta, \theta$, to refer to constituents of feature graphs.
- When displaying feature graphs, the root is depicted as a grey-colored node, usually at the top or the left side of the graph.
- The identities of the nodes are arbitrary, and we use generic names such as q_0, q_1 etc. to refer to them.

Feature graphs

Example (Feature graphs)

In the following graph, the leaves q_2 and q_3 bear no marking; in other words, the marking function θ is undefined for the two sinks in its domain.



The graph displayed above is $\langle Q, \bar{q}, \delta, \theta \rangle$, where $Q = \{q_0, q_1, q_2, q_3\}$, $\bar{q} = q_0$, $\delta(q_0, \text{AGR}) = q_1$, $\delta(q_1, \text{NUM}) = q_2$, $\delta(q_1, \text{PERS}) = q_3$, $Q_S = \{q_2, q_3\}$, and θ is undefined for its entire domain.

Feature graphs

- A feature graph is **empty** if it consists of a single unmarked node with no arcs.
- A feature graph is **atomic** if it consists of a single marked node with no arcs.

Empty and atomic feature graphs

Example (Empty and atomic feature graphs)

A, an empty feature graph: q_0

B, an atomic feature graph: q_0 $_{pl}$

Paths

- The concept of *paths* is natural when graphs are concerned.
- A **path** (over $\textsc{Feats}$) is a finite sequence of features, and the set $\textsc{Paths} = \textsc{Feats}^*$ is the collection of all paths.
- Meta-variables π, α (with or without subscripts) range over paths.
- ϵ is the empty path, denoted also by '$\langle\rangle$'.
- The *length* of a path π is denoted $|\pi|$.
- For example, if $\textsc{Feats} = \{\textsc{a}, \textsc{b}\}$ then $\textsc{Paths}$ includes $\epsilon, \langle\textsc{a}\rangle, \langle\textsc{b}\rangle, \langle\textsc{a},\textsc{b},\textsc{a}\rangle, \langle\textsc{b},\textsc{b},\textsc{b},\textsc{b},\textsc{a},\textsc{b}\rangle$, etc.
- While a path is a purely syntactic notion (every sequence of features constitutes a path), interesting paths are those that can be interpreted as actual paths in some graph, leading from the root to some node.

Paths

- The definition of δ is therefore extended to paths: given a feature graph $A = \langle Q_A, \bar{q}_A, \delta_A, \theta_A \rangle$, define $\hat{\delta}_A : Q_A \times \textsc{Paths} \rightarrow Q_A$ as follows:

$$\hat{\delta}_A(q, \epsilon) = q$$
$$\hat{\delta}_A(q, f\pi) = \hat{\delta}_A(\delta_A(q, f), \pi) \quad \text{(defined only if } \delta_A(q, f)\downarrow)$$

- Since for every node $q \in Q_A$ and every feature $\textsc{f} \in \textsc{Feats}$, $\delta_A(q, \textsc{f}) = \hat{\delta}_A(q, \langle \textsc{f} \rangle)$, we identify $\hat{\delta}$ with δ in the future and use only the latter. When the index (A) is clear from the context, it is omitted. When $\delta_A(q, \pi) = q'$ we say that π **leads** (in A) from q to q'.

Paths

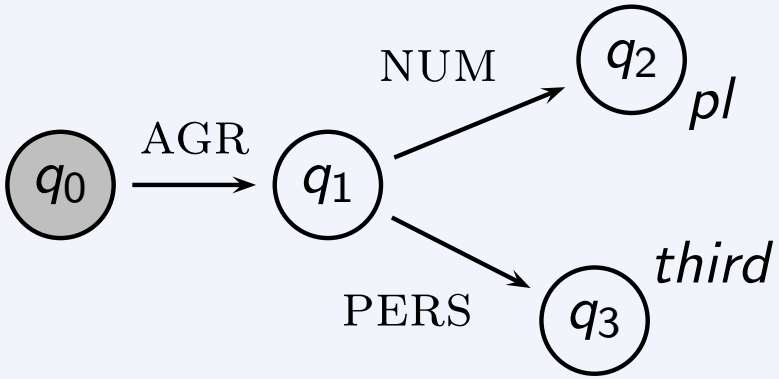
Definition (Paths)

The **paths** of a feature graph A are $\Pi(A) = \{\pi \in \textsc{Paths} \mid \delta_A(\bar{q}_A, \pi)\downarrow\}$.

Paths

Example (Paths)

Consider the following feature graph, A:



Its paths are

$$\Pi(A) = \{\epsilon, \langle \text{AGR} \rangle, \langle \text{AGR NUM} \rangle, \langle \text{AGR PERS} \rangle\}$$

Path values

- Of particular interest are paths which lead from the root of a feature graph to some node in the graph.
- For such paths we define the notion of a *value*, which is the sub-graph whose root is the node at the end of the path.
- It would have been possible to define as value the node itslef, rather than the sub-graph is induces; the choice is a matter of taste, as moving from one view of values to another is trivial.

Path values

Definition (Path value)

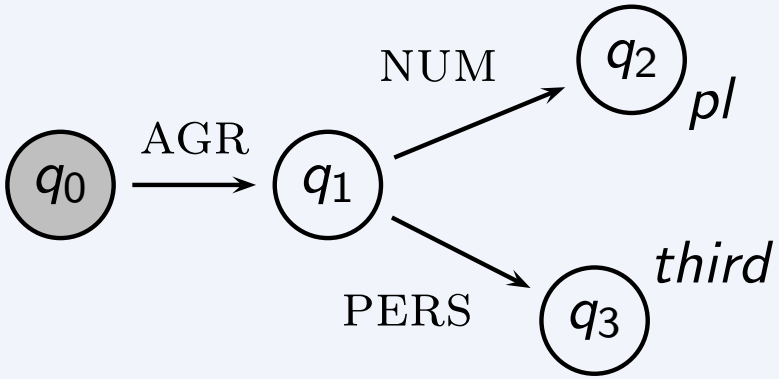
For a feature graph $A = \langle Q_A, \bar{q}_A, \delta_A, \theta_A \rangle$ and a path $\pi \in \Pi(A)$, the **value** $val_A(\pi)$ of π in A is a feature graph $B = \langle Q_B, \bar{q}_B, \delta_B, \theta_B \rangle$, over the same signature as A, where:

- $\bar{q}_B = \delta_A(\bar{q}_A, \pi)$
- $Q_B = \{q' \in Q_A \mid \text{for some } \pi',\ \delta_A(\bar{q}_B, \pi') = q'\}$ (Q_B is the set of nodes reachable from $\bar{q}_B$)
- for every feature f and for every $q' \in Q_B$, $\delta_B(q', \mathrm{F}) = \delta_A(q', f)$ (δ_B is the restriction of δ_A to Q_B)
- for every $q' \in Q_B$, $\theta_B(q') = \theta_A(q')$ (θ_B is the restriction of θ_A to Q_B)

Paths

Example (Paths)

Consider the following feature graph, A:



Its paths are

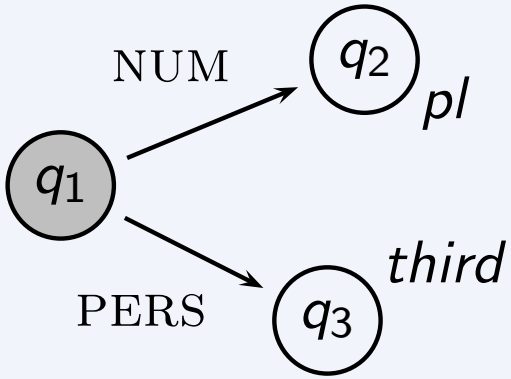
$$\Pi(A) = \{\epsilon, \langle \text{AGR} \rangle, \langle \text{AGR NUM} \rangle, \langle \text{AGR PERS} \rangle\}$$

Path values

Example (Path values)

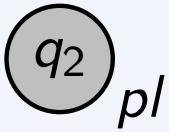
The value of the path $\langle \text{AGR} \rangle$ in A is:

$val_A(\langle \text{AGR} \rangle) =$



and the value of the path $\langle \text{AGR NUM} \rangle$ in A is:

$val_A(\langle \text{AGR NUM} \rangle) =$



Note that, for example, the value of $\langle \text{AGR PERS NUM} \rangle$ in A is undefined.

Reentrancy

- The definition of path values raises the question of when two paths have equal values.
- We distinguish between paths which lead to one and the same node, and those whose values are isomorphic but not identical.
- The former case is called *reentrancy*.

Reentrancy

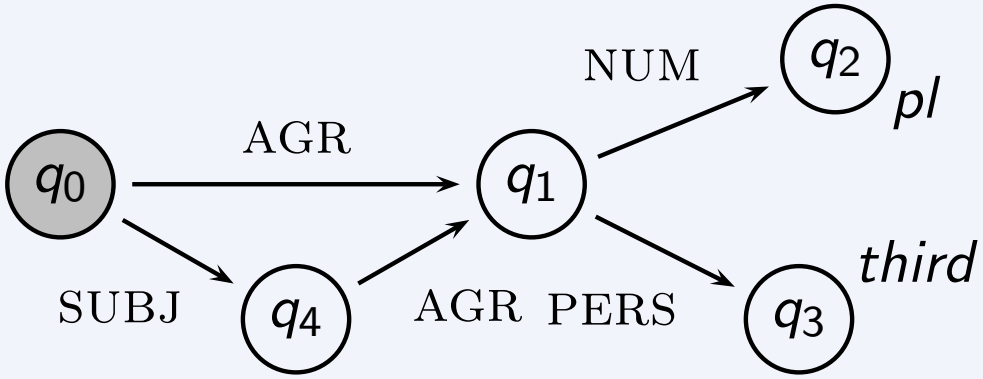
Definition (Reentrancy)

Let $A = \langle Q, \bar{q}, \delta, \theta \rangle$ be a feature graph. Two paths $\pi_1, \pi_2 \in \Pi(A)$ are **reentrant** in A, denoted $\pi_1 \overset{A}{\leftrightsquigarrow} \pi_2$, iff $\delta(\bar{q}, \pi_1) = \delta(\bar{q}, \pi_2)$, implying $val_A(\pi_1) = val_A(\pi_2)$. A feature graph A is **reentrant** iff there exist two *distinct* paths $\pi_1, \pi_2 \in \Pi(A)$ such that $\pi_1 \overset{A}{\leftrightsquigarrow} \pi_2$.

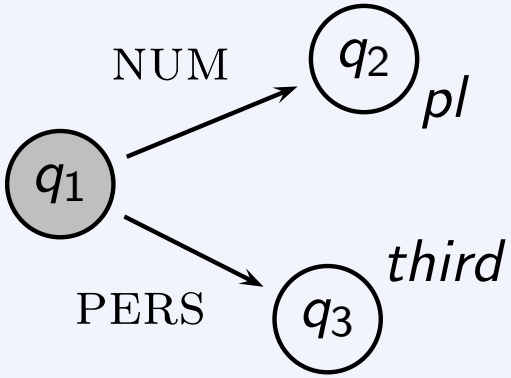
Reentrancy

Example (A reentrant feature graph)

This feature graph, A, is reentrant because $\delta_A(q_0, \langle \text{AGR} \rangle) = \delta_A(q_0, \langle \text{SUBJ,AGR} \rangle)$



The (single) value of the (different) paths $\langle \text{AGR} \rangle$ and $\langle \text{SUBJ AGR} \rangle$ in A is:



Reentrancy

- The notion of reentrancy touches on the issue of the distinction between *type-* and *token-identity*.
- Two feature graphs are token identical if their components (i.e., their sets of nodes, roots, transition functions and atom marking functions) are identical.
- They are type-identical if they are isomorphic, not necessarily requiring their nodes to be identical.
- We will discuss feature graph isomorphism later.

Cicles

- Early feature structure based formalisms used to employ only *acyclic* feature graphs.
- However, modern ones usually allow (or even require) feature structures to be possibly *cyclic*.
- While the linguistic motivation for cyclic feature structures is limited, there is good practical motivation for allowing them: when implementing a system for manipulating feature graphs, it is usually easier to support cycles than to guarantee that all the graphs in a system are acyclic.
- The reason is that unification, which is the major operation defined on feature graphs, can yield a cyclic graph even when its operands are acyclic.

Cicles

Definition (Cycles)

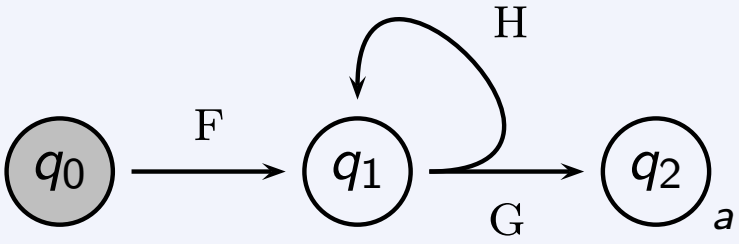
A feature graph $A = \langle Q_A, \bar{q}_A, \delta_A, \theta_A \rangle$ is **cyclic** if two paths $\pi_1, \pi_2 \in \Pi(A)$, where π_1 is a proper subsequence of π_2, are reentrant: $\pi_1 \overset{A}{\leftrightsquigarrow} \pi_2$. A is **acyclic** otherwise.

- Note that cyclicity is a special case of reentrancy (every cyclic feature graph is reentrant, but not vice versa).
- A corollary of the definition is that when a feature graph is cyclic, it has at least one node q such that $\delta(q, \alpha) = q$ for some non-empty path α.

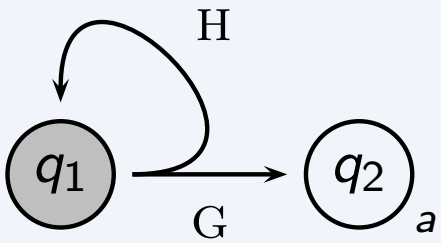
Cicles

Example (A cyclic feature graph)

Following is a cyclic feature graph, C:



The value of the path $\langle \text{F} \rangle$ in C, as well as the values of the (infinitely many) paths $\langle \text{F H}^n \rangle$, for $n \geq 0$, is the same feature graph:



Feature graph isomorphism

- Since feature graphs are just a special case of directed, labeled graphs, we can adapt the well-defined notion of graph *isomorphism* to feature graphs.
- Informally, two graphs are isomorphic when they have the same structure; the identites of their nodes may differ without affecting the structure.
- In our case, we require also that the labels of sink nodes be identical in order for two graphs to be considered isomorphic.

Feature graph isomorphism

Definition (Feature graph isomorphism)

Two feature graphs $A = \langle Q_A, \bar{q}_A, \delta_A, \theta_A \rangle$ and $B = \langle Q_B, \bar{q}_B, \delta_B, \theta_B \rangle$ are **isomorphic**, denoted $A \sim B$, iff there exists a one-to-one and onto mapping $i : Q_A \rightarrow Q_B$, called an **isomorphism**, such that:

- $i(\bar{q}_A) = \bar{q}_B$;
- for all $q_1, q_2 \in Q_A$ and $f \in \textsc{Feats}$, $\delta_A(q_1, f) = q_2$ iff $\delta_B(i(q_1), f) = i(q_2)$; and
- for all $q \in Q_A$, $\theta_A(q) = \theta_B(i(q))$ (either both are undefined, or both are defined and equal).

Feature graph subsumption

Definition (Subsumption)

Let $A_1 = \langle Q_1, \bar{q}_1, \delta_1, \theta_1 \rangle$ and $A_2 = \langle Q_2, \bar{q}_2, \delta_2, \theta_2 \rangle$ be two feature graphs. A_1 **subsumes** A_2 (denoted by $A_1 \sqsubseteq A_2$) iff there exists a total function $h : Q_1 \rightarrow Q_2$, called a **subsumption morphism**, such that

- $h(\bar{q}_1) = \bar{q}_2$
- for every $q \in Q_1$ and for every f such that $\delta_1(q, f)\downarrow$, $h(\delta_1(q, f)) = \delta_2(h(q), f)$
- for every $q \in Q_1$, if $\theta_1(q)\downarrow$ then $\theta_1(q) = \theta_2(h(q))$.

If $A_1 \sqsubseteq A_2$ then A_1 is said to *subsume*, or be *more general than* A_2; A_2 is *subsumed by*, or is *more specific than*, A_1.

Subsumption

- The morphism h associates with every node in Q_1 a node in Q_2; if an arc labeled f connects q with q', then such an arc connects $h(q)$ with $h(q')$.
- In other words, δ and h *commute*, as depicted in the following diagram, where δ-arcs are depicted using solid lines, whereas h-mappings are depicted using dashed lines:

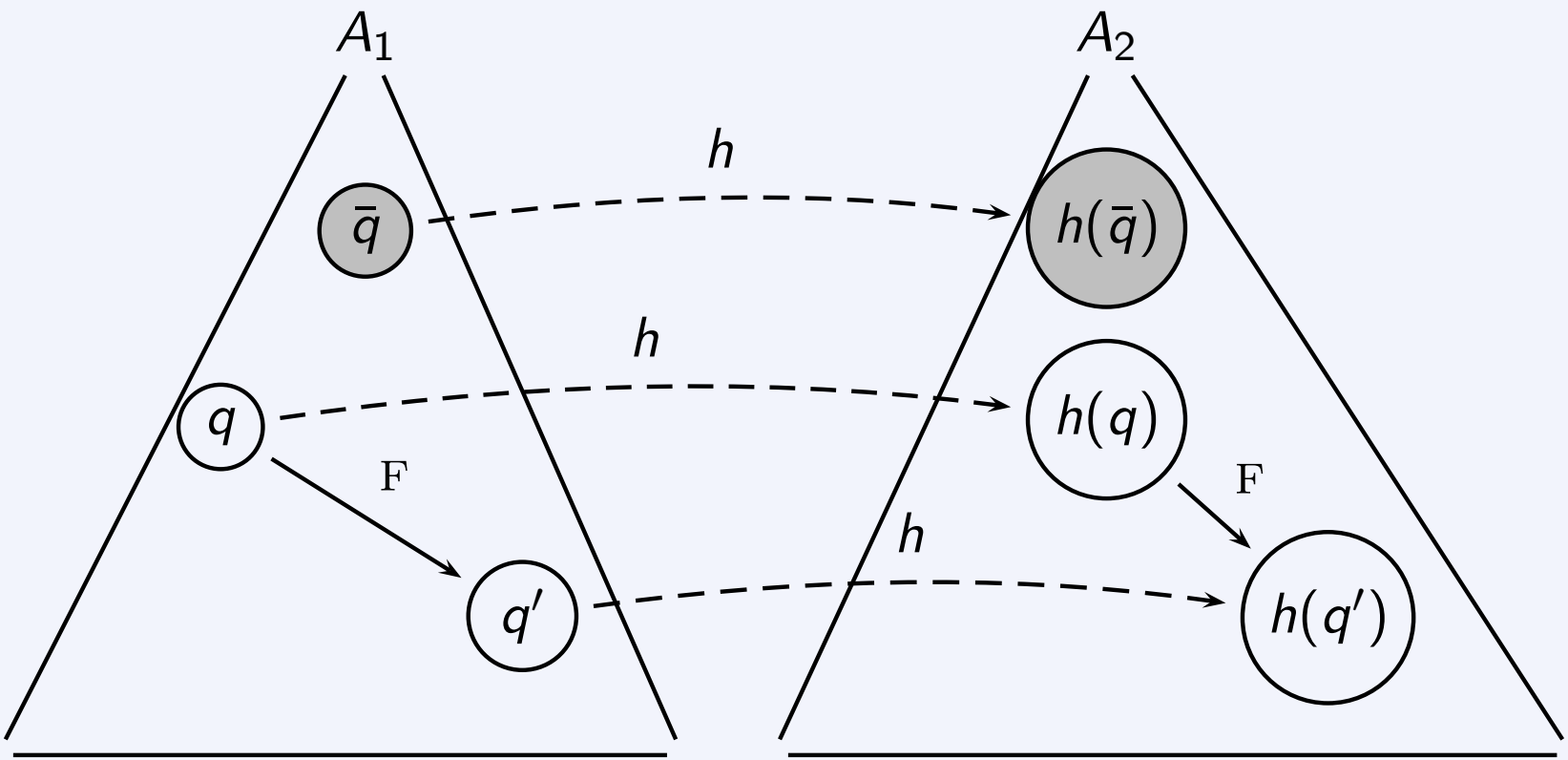
$$\begin{tikzcd}
 & \bigcirc \arrow[r, dashed, "h"] \arrow[d, "f"] & \bigcirc \arrow[d, "f"] \\
\delta : & \bigcirc \arrow[r, dashed, "h"'] & \bigcirc
\end{tikzcd}$$

Subsumption

- In addition, if a node $q \in Q_1$ is marked by an atom, then its image $h(q)$ must be marked by the same atom (recall that only sinks can be thus marked).
- Note that if a sink in Q_1 is *not* marked, there is no constraint on its image (in particular, it can be a non-sink).

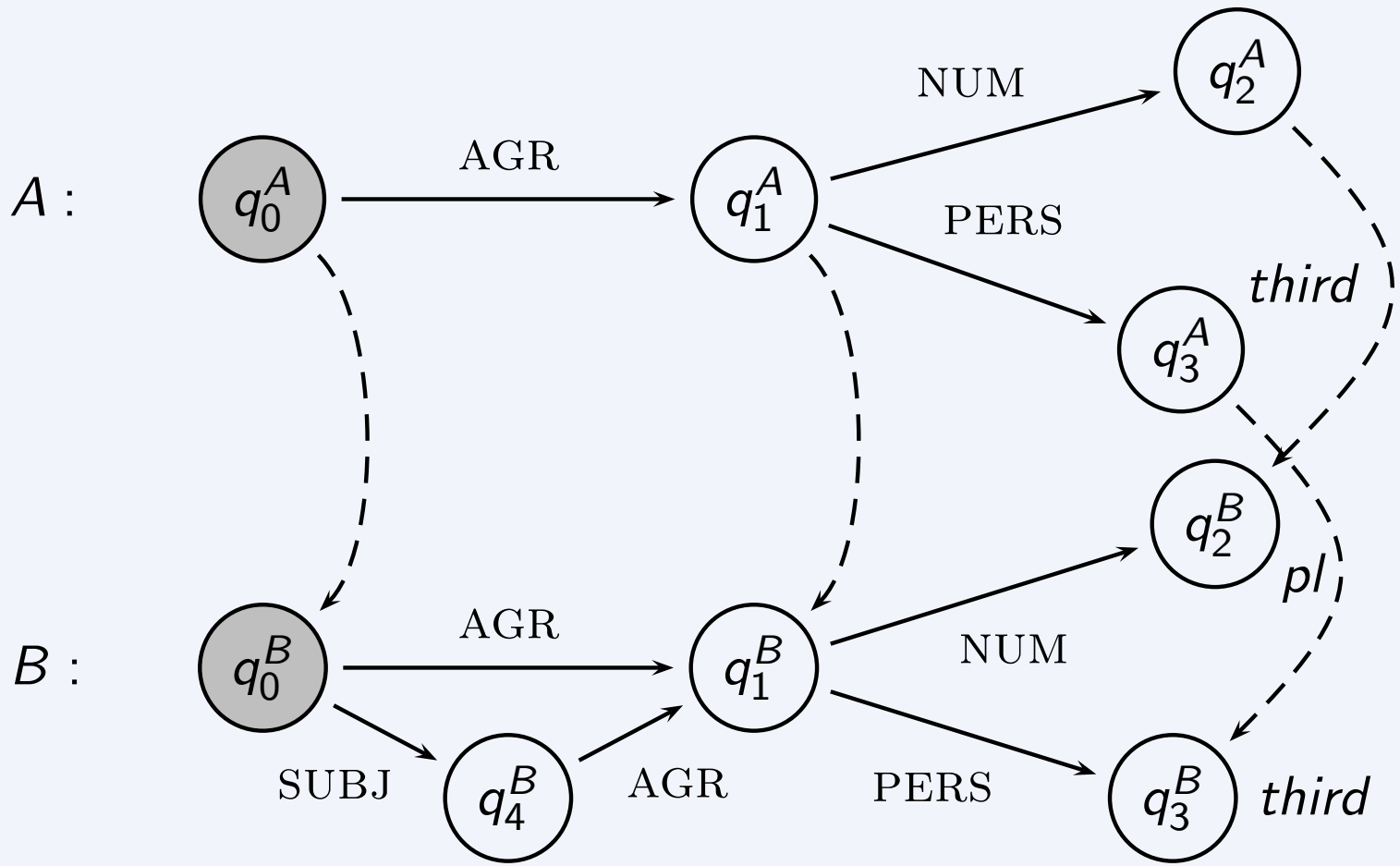
Subsumption morphism

Example (Subsumption morphism)



Subsumption morphism

Example (Subsumption)



Subsumption morphism

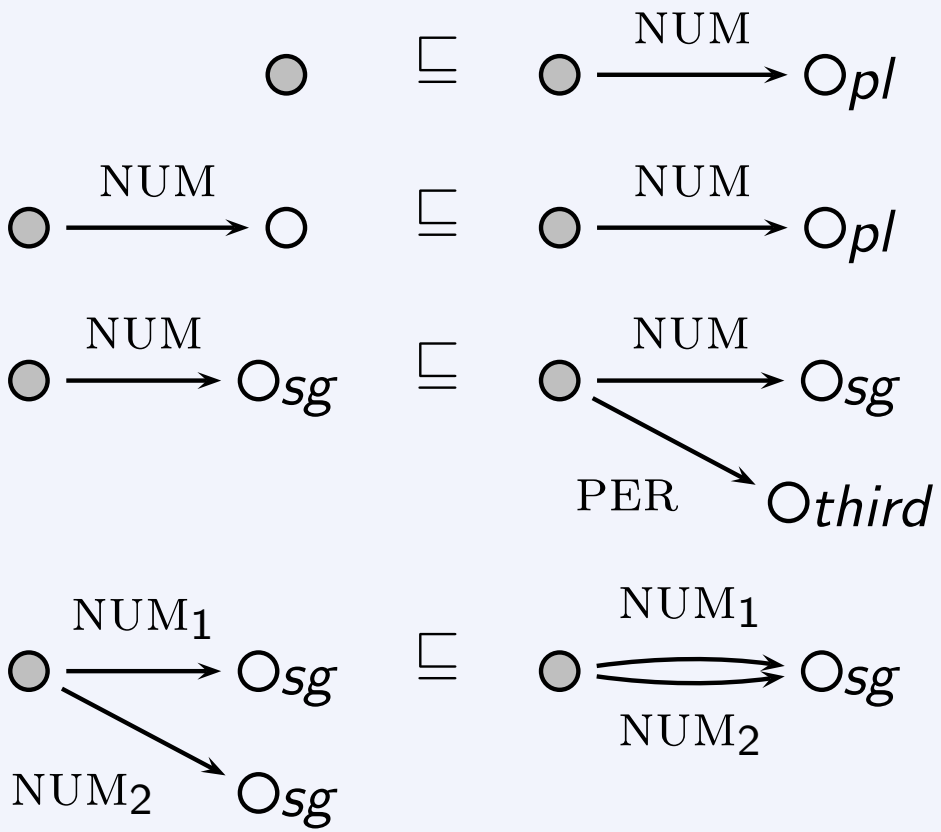
- Indeed, B can—and does—have nodes that do not correspond to nodes in A: such is q_4^B in the example.
- In addition, while the sink q_2^A is not marked by an atom (that is, it is a variable), its image in B, q_2^B, is marked as pl.
- Notice that no subsumption morphism can be defined from Q_B to Q_A, since there is no node into which q_4^B can be mapped.
- In particular, it cannot be mapped to the root of A since this would necessitate an arc from q_0^A to itself (as the root of A would be the image of both q_4^B and q_0^B).
- Trying to take h^{-1} as an inverse subsumption morphism will fail both because of q_4^B and because it would map q_2^B to q_2^A, violating the last clause of the subsumption relation (a marked sink must be mapped to a sink with the same mark).
- We conclude that $B \not\sqsubseteq A$.

Subsumption

- Given a feature structure, what modifications can be made to it in order for it to become more specific? Three different kinds of modifications are possible:
  1. Adding arcs;
  2. Adding reentrancies;
  3. Marking unmarked sinks by some atom.

Subsumption

Example (Subsumption as an order on information)



adding arcs

adding atomic marks

adding arcs

adding reentrancies

Subsumption

Lemma

If $A \sqsubseteq B$ *then* $\Pi(A) \subseteq \Pi(B)$.

Subsumption

Lemma

If $A \sqsubseteq B$ *then for each* $\pi \in \Pi(A)$*, if* $\theta_A(\delta_A(\bar{q}_A, \pi))\downarrow$ *then* $\theta_B(\delta_B(\bar{q}_B, \pi))\downarrow$ *and* $\theta_A(\delta_A(\bar{q}_A, \pi)) = \theta_B(\delta_B(\bar{q}_B, \pi))$.

Subsumption

Lemma

If $A \sqsubseteq B$ *and* π_1, π_2 *are reentrant in* A *(that is,* $\pi_1 \overset{A}{\leftrightsquigarrow} \pi_2$*) then* π_1, π_2 *are reentrant in* B *(that is,* $\pi_1 \overset{B}{\leftrightsquigarrow} \pi_2$*).*

Subsumption

Corollary

If $A \sqsubseteq B$, then:

- $\Pi(A) \subseteq \Pi(B)$
- *for each $\pi \in \Pi(A)$, if $\theta_A(\delta_A(\bar{q}_A, \pi))\downarrow$ then $\theta_B(\delta_B(\bar{q}_B, \pi))\downarrow$ and $\theta_A(\delta_A(\bar{q}_A, \pi)) = \theta_B(\delta_B(\bar{q}_B, \pi))$*
- *for each $\pi_1, \pi_2 \in \Pi(A)$, if $\pi_1 \overset{A}{\leftrightsquigarrow} \pi_2$ then $\pi_1 \overset{B}{\leftrightsquigarrow} \pi_2$ (and, therefore, if A is reentrant/cyclic then so is B).*

Subsumption

Theorem

If A is an atomic feature graph and $A \sqsubseteq B$, then $A \sim B$.

Subsumption

Theorem

Subsumption has a least element: there exists a feature graph A such that for all feature graph B, $A \sqsubseteq B$.

Proof.

Consider the (empty) feature graph $A = \langle \{q_0\}, q_0, \delta, \theta \rangle$, where δ and θ are undefined for their entire domains. For every feature graph B, $A \sqsubseteq B$ by mapping (through h) the root q_0 to the root of B, $\bar{q}_B$. The two clauses of the definition of subsumption hold vacuously. □

Subsumption

Theorem

Subsumption is reflexive: for every feature graph A, $A \sqsubseteq A$.

Proof.

Take h to be the identity function that maps every node in A to itself. □

Subsumption

Theorem

Subsumption is transitive: if $A \sqsubseteq B$ and $B \sqsubseteq C$ then $A \sqsubseteq C$.

Subsumption

Theorem

Subsumption is not antisymmetric: if $A \sqsubseteq B$ and $B \sqsubseteq A$ then not necessarily $A = B$.

Proof.

Consider the feature graphs $A = \langle \{\bar{q}_A\}, \bar{q}_A, \delta, \theta \rangle$ and $B = \langle \{\bar{q}_B\}, \bar{q}_B, \delta, \theta \rangle$, where δ and θ are undefined for their entire domains, and where $\bar{q}_A \neq \bar{q}_B$. Trivially, both $A \sqsubseteq B$ and $B \sqsubseteq A$, but $A \neq B$. $\square$

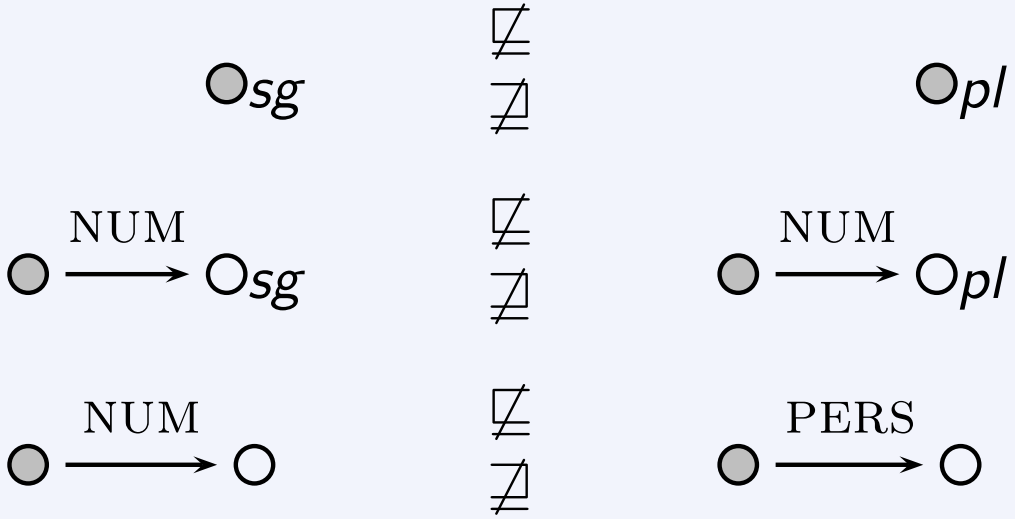
Subsumption

- Thus, feature graph subsumption forms a partial pre-order on feature graphs.
- It is a *pre*-order since it is not antisymmetric; it is *partial* as there are feature graphs that are incomparable with respect to subsumption.

Subsumption

Example (Feature graph subsumption is a partial relation)

Feature graphs can be incomparable due to inconsistency (contradicting information) or to complementary information.



Subsumption

There is a clear connection between feature graph isomorphism and feature graph subsumption:

Theorem

$A \sim B$ iff $A \sqsubseteq B$ and $B \sqsubseteq A$.

AVMs

- We now return to attribute-value matrices (AVMs).
- This is the view that we will adopt for depicting feature structures (and grammars based on them), both because they are easy to present on paper and because of their centrality in existing literature.
- Like feature graphs, AVMs are defined over a signature of features and atoms, which we fix below.
- In addition, AVMs make use of *variables*, also called *tags* below. Meta-variables X, Y, Z, etc. range over over variables.
- Variables are used to encode sharing of values, as will be clear presently.
- When AVMs are concerned, we follow the convention of the linguistic literature by which variables are natural numbers, depicted in boxes, e.g., $\boxed{3}$.

AVMs

Definition (AVMs)

Given a signature $\mathcal{S}$, the set $\textsc{Avms}(\mathcal{S})$ of **AVMs** over $\mathcal{S}$ is the least set satisfying the following two clauses:

1. $M = Xa \in \textsc{Avms}(\mathcal{S})$ for any $a \in \textsc{Atoms}$ and $X \in \textsc{Tags}$; M is said to be **atomic** and X is the **tag** of M, denoted $tag(M) = X$.
2. $M = X[f_1 : M_1, \ldots, f_n : M_n] \in \textsc{Avms}(\mathcal{S})$ for $n \geq 0$, $X \in \textsc{Tags}$, $f_1, \ldots, f_n \in \textsc{Feats}$ and $M_1, \ldots, M_n \in \textsc{Avms}(\mathcal{S})$, where $f_i \neq f_j$ if $i \neq j$. M is said to be **complex**, and X is the **tag** of M, denoted $tag(M) = X$. If $n = 0$, $M = X[]$ is an **empty AVM**.

Note that two AVMs which differ only in their tag are distinct: if $X \neq Y$, $X\left[\cdots\right] \neq Y\left[\cdots\right]$. In particular, there is no *unique* empty AVM. Note also that the same variable can be used more than once in an AVM.

AVMs

Example (AVMs)

Consider a signature consisting of $\textsc{Atoms} = \{a\}$ and $\textsc{Feats} = \{\textsc{f},\textsc{g}\}$. Then $M_1 = \boxed{4}a$ is an AVM by the first clause of the definition, $M_2 = \boxed{2}[\,]$ is an empty AVM by the second clause, $M_3 = \boxed{3}\begin{bmatrix}\textsc{f}: & \boxed{4}a\end{bmatrix}$ is an AVM by the second clause (using M_1 as the value of $\textsc{f}$, so that $fval(M_3, \textsc{f}) = M_1$), and

$$M_4 = \boxed{2}\begin{bmatrix}\textsc{g}: & \boxed{3}\begin{bmatrix}\textsc{f}: & \boxed{4}a\end{bmatrix} \\ \textsc{f}: & \boxed{2}[\,]\end{bmatrix}$$

is an AVM by the second clause, as is

$$M_5 = \boxed{4}\begin{bmatrix}\textsc{g}: & \boxed{3}\begin{bmatrix}\textsc{f}: & \boxed{4}a\end{bmatrix} \\ \textsc{f}: & \boxed{2}[\,]\end{bmatrix}$$

AVMs

- Meta-variables M, with or without subscripts, range over $\textsc{Avms}$; the parameter $\mathcal{S}$ is omitted when it is clear from the context.
- The *domain* of an AVM M, denoted $dom(M)$, is undefined when M is atomic, and $\{f_1, \ldots, f_n\}$ when M is complex (hence, $dom(M)$ is empty for an empty AVM).
- The *value* of some feature $f \in \textsc{Feats}$ in M, denoted $fval(M, f)$, is defined if $f = f_i \in dom(M)$, in which case it is M_i, and undefined otherwise.

Sub-AVMs

Definition (Sub-AVMs)

Given an AVM M, its **sub-AVMs** are $SubAVM(M)$, defined as:

1. $SubAVM(Xa) = \{Xa\}$
2. $SubAVM(X[f_1 : M_1, \ldots, f_n : M_n]) = X[f_1 : M_1, \ldots, f_n : M_n] \bigcup \cup_{1 \leq i \leq n} SubAVM(M_i)$

AVMs

Definition (Tags)

Given an AVM M, its **tags** $Tags(M)$ are defined as:

1. $Tags(Xa) = \{X\}$
2. $Tags(X[f_1 : M_1, \ldots, f_n : M_n]) = X \cup_{1 \leq i \leq n} Tags(M_i)$

Definition (Tagset)

The **tagset** of an AVM M and a tag $X \in Tags(M)$ is the set of sub-AVMs of M (including M itself) which are tagged by X:
$TagSet(M, X) = \{M' \in SubAVM(M) \mid tag(M') = X\}$.

AVMs

Example (AVMs)

Let:

$$M_4 = \boxed{2}\begin{bmatrix} \mathrm{G}: & \boxed{3}\begin{bmatrix} \mathrm{F}: & \boxed{4}a \end{bmatrix} \\ \mathrm{F}: & \boxed{2}[\,] \end{bmatrix}$$

$fval(M_4, \mathrm{F}) = \boxed{2}[\,]$. Observe that $Tags(M_4) = \{\boxed{2}, \boxed{3}, \boxed{4}\}$. Also, $TagSet(M_4, \boxed{4})$ is $\{\boxed{4}a\}$, $TagSet(M_4, \boxed{3})$ is $\{\boxed{3}\begin{bmatrix} \mathrm{F}: & \boxed{4}a \end{bmatrix}\}$ and $TagSet(M_4, \boxed{2})$ is $\{M_4, \boxed{2}[\,]\}$.
Trivially, $tag(M_4) = \boxed{2}$.

AVMs

Example (AVMs)

Let:

$$M_5 = \boxed{4}\begin{bmatrix} \mathrm{G}: & \boxed{3}\begin{bmatrix}\mathrm{F}: & \boxed{4}a\end{bmatrix} \\ \mathrm{F}: & \boxed{2}[\,] \end{bmatrix}$$

Similarly, $fval(M_5, \mathrm{F}) = \boxed{2}[\,]$, whereas $fval(M_5, \mathrm{G}) = \boxed{3}\begin{bmatrix}\mathrm{F}: & \boxed{4}a\end{bmatrix}$. Observe that $Tags(M_5) = \{\boxed{2}, \boxed{3}, \boxed{4}\}$.Also $TagSet(M_5, \boxed{2}) = \{\boxed{2}[\,]\}$, $TagSet(M_5, \boxed{3}) = \{\boxed{3}\begin{bmatrix}\mathrm{F}: & \boxed{4}a\end{bmatrix}\}$ and $TagSet(M_5, \boxed{4}) = \{M_5, \boxed{4}a\}$. Trivially, $tag(M_5) = \boxed{4}$.

AVMs

Example (AVMs)

As another example, consider the AVM

$$M_6 = \boxed{1}\begin{bmatrix}\text{F}: & \boxed{1}\begin{bmatrix}\text{F}: & \boxed{1}\begin{bmatrix}\text{F}: & \boxed{1}[\,]\end{bmatrix}\end{bmatrix}\end{bmatrix}$$

Here, $Tags(M_6) = \{\boxed{1}\}$, and $TagSet(M_6, \boxed{1})$ is:

$$\{M_6, \boxed{1}\begin{bmatrix}\text{F}: & \boxed{1}\begin{bmatrix}\text{F}: & \boxed{1}[\,]\end{bmatrix}\end{bmatrix}, \boxed{1}\begin{bmatrix}\text{F}: & \boxed{1}[\,]\end{bmatrix}, \boxed{1}[\,]\}$$

Of course, $tag(M_6) = \boxed{1}$.

Well-formed AVMs

- Consider some AVM $M = \boxed{1}\begin{bmatrix} f_1 : & \boxed{2}M_1 \\ f_2 : & \boxed{2}M_2 \end{bmatrix}$ where $M_1 \neq M_2$.
- Both M_1 and M_2 are sub-AVMs of M, and both have the same tag, although they are different.
- In other words, the recursive definition of AVMs allows two different, contradicting AVMs to be in the *TagSet* of the same variable.
- To eliminate such cases, we define well-formed AVMs as follows:

Definition (Well-formed AVMs)

An AVM M is **well-formed** iff for every variable $X \in Tags(M)$, $TagSet(M, X)$ includes at most one non-empty AVM.

Reentrancy

Example (A reentrant AVM)

The following AVM is reentrant but not cyclic:

$$\boxed{0}\begin{bmatrix} \text{AGR}: & \boxed{1}\begin{bmatrix} \text{NUM}: & \boxed{2}\textit{pl} \\ \text{PERS}: & \boxed{3}\textit{third} \end{bmatrix} \\ \text{SUBJ}: & \boxed{4}\begin{bmatrix} \text{AGR}: & \boxed{1} \end{bmatrix} \end{bmatrix}$$

Conventions

- We introduce three conventions regarding the depiction of *well-formed* AVMs, motivated by the fact that variables are used primarily to indicate value sharing.
- If a variable occurs more than once then its value is explicated only once; where this value is explicated (i.e., next to which occurrence of the variable) is immaterial.
- Variables which occur only once can be omitted.
- The empty AVM is sometimes omitted when it is associated with a variable.
- The first convention is crucial in the case of cyclic AVMS: there is no finite representation of cyclic AVMs unless this convention is adopted.

Conventions

Example (Shorthand notation for AVMs)

Consider the following AVM:

$$\boxed{6}\begin{bmatrix} \text{F}: & \boxed{3}\begin{bmatrix} \end{bmatrix} \\ \text{G}: & \boxed{4}\begin{bmatrix} \text{H}: & \boxed{3}a \end{bmatrix} \\ \text{H}: & \boxed{2}\begin{bmatrix} \end{bmatrix} \end{bmatrix}$$

Notice that it is well-formed, since the only variable occurring more than once ($\boxed{3}$) is associated with a non-empty value (a) only once.

- We can therefore leave only one occurrence of the value explicit
- The tag $\boxed{2}$ is associated with the empty feature structure, which can be omitted
- Finally, the tags $\boxed{4}$ and $\boxed{6}$ occur only once, so they can be omitted
- This is the conventional form of the AVM.

AVM equivalence

Example (AVM equivalence)

M_1 and M_2 differ only in the instance of $\boxed{0}$ whose value is explicated:

$$M_1 = \boxed{0}\begin{bmatrix} \text{AGR}: & \boxed{1}\begin{bmatrix} \text{NUM}: & \boxed{2}pl \\ \text{PERS}: & \boxed{3}third \end{bmatrix} \\ \text{SUBJ}: & \boxed{4}\begin{bmatrix} \text{AGR}: & \boxed{1} \end{bmatrix} \end{bmatrix}$$

$$M_2 = \boxed{0}\begin{bmatrix} \text{AGR}: & \boxed{1} \\ \text{SUBJ}: & \boxed{4}\begin{bmatrix} \text{AGR}: & \boxed{1}\begin{bmatrix} \text{NUM}: & \boxed{2}pl \\ \text{PERS}: & \boxed{3}third \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

Then $M_1 \preceq M_2$ and $M_2 \preceq M_1$.

AVM equivalence

Example (AVM renamings)

The following two AVMs are renamings of each other:

$$M_1 = \boxed{0}\begin{bmatrix} \text{AGR}: & \boxed{1}\begin{bmatrix} \text{NUM}: & \boxed{2}\textit{pl} \\ \text{PERS}: & \boxed{3}\textit{third} \end{bmatrix} \\ \text{SUBJ}: & \boxed{4}\begin{bmatrix} \text{AGR}: & \boxed{1} \end{bmatrix} \end{bmatrix}$$

$$M_2 = \boxed{10}\begin{bmatrix} \text{AGR}: & \boxed{11} \\ \text{SUBJ}: & \boxed{14}\begin{bmatrix} \text{AGR}: & \boxed{11}\begin{bmatrix} \text{NUM}: & \boxed{12}\textit{pl} \\ \text{PERS}: & \boxed{13}\textit{third} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The correspondence between feature graphs and AVMs

- AVMs are the entities that the linguistic literature employs to depict feature structures;
- feature graphs are well-understood mathematical entities to which various results of graph theory can be applied.
- We define the relationship between these two views.

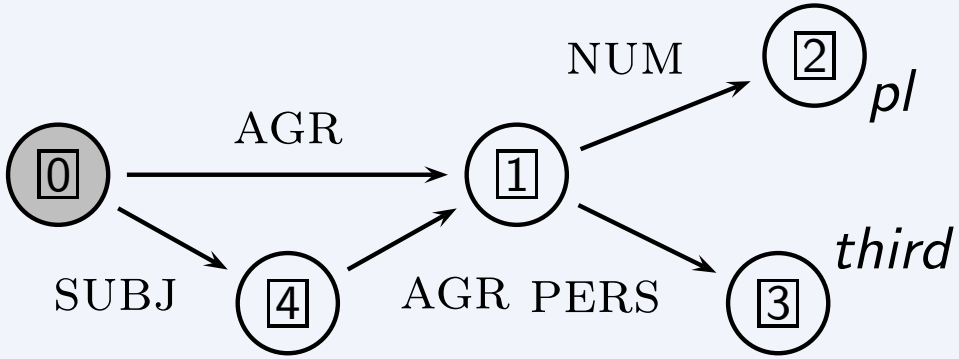
From AVMs to feature graphs

Example (AVM to graph mapping)

A reentrant AVM and its feature graph image:

$$M = \boxed{0}\begin{bmatrix} \text{AGR}: & \boxed{1}\begin{bmatrix} \text{NUM}: & \boxed{2}\textit{pl} \\ \text{PERS}: & \boxed{3}\textit{third} \end{bmatrix} \\ \text{SUBJ}: & \boxed{4}\begin{bmatrix} \text{AGR}: & \boxed{1} \end{bmatrix} \end{bmatrix}$$

$\phi(M) =$



Feature structures in a broader context

- Feature structures are utilized by many grammatical formalisms to encode different kinds of linguistic information: they serve in representing phonological, morphological, syntactic and semantic knowledge.
- But the use of feature structures is not limited to computational linguistics; indeed, they are present in other areas of computer science as well.

Feature structures in a broader context

- A somewhat degenerate form of feature structures is utilized by many programming languages: *records* (as in Pascal, known as *structures* in C).
- There are some major differences between records and feature structures.
  - The notion of sharing that is central to feature structures is less significant for records.
  - The values of record fields are not necessarily other records – different data types can be freely used; hence transfer of values is mediated through explicit assignments, not unifications.
  - Unification-based formalisms usually do not allow such a diversity of operations to apply to feature structures as programming languages allow to records.
  - In particular, arithmetic operations are usually not applicable to feature structures' values, while they are very natural to numeric records' fields.

Feature structures in a broader context

- Logic programming languages such as Prolog manipulate *first-order terms* (FOTs), which might be viewed as a special case of feature structures.
- There are some important differences between feature structures and FOTs.
  - FOTs are essentially trees, with possibly shared leaves, whereas feature structures allow reentrancies to occur in every level of the structure. Feature structures can be cyclic, in contrast to (ordinary) FOTs.
  - FOTs use positional encoding of argument structures, with no features.
  - Two FOTs are unifiable only if they have the same functor and the same arity, while two feature structures might be unifiable even if they have different number of features.

Unification

- The subsumption relation *compares* the information content of feature structures.
- Unification *combines* the information that is contained in two (compatible) feature structures.
- We use the term 'unification' to refer to both the operation and its result. Whenever two feature structures are related, they are assumed to be over the *same* signature.

Unification

- The mathematical interpretation of "combining" two members of a partially ordered set is to take the *least upper bound* of the two operands with respect to the partial order; in our case, subsumption.
- Indeed, feature structure unification is exactly that.
- However, since subsumption is antisymmetric for feature structures and AFSs but not for feature graphs and AVMs, a unique least upper bound cannot be guaranteed for all four views.

Feature structure unification

Definition (Feature structure unification)

Two feature structures fs_1 and fs_2 are **consistent** if they have an upper bound (with respect to subsumption), and **inconsistent** otherwise. If fs_1 and fs_2 are consistent, their **unification**, denoted $fs_1 \hat{\sqcup} fs_2$, is their least upper bound with respect to subsumption.

If two feature structures have an upper bound, they have a (unique) least upper bound.

Feature graph unification

- While the definition of unification as least upper bound is useful mathematically, it does not tells us *how* to compute the unification of two given feature structures.
- To this end, we provide a constructive definition in terms of feature graphs, which induces an algorithm for computing unification.
- For reasons that will be clear presently, we require that the two feature graphs be node-disjoint.

Feature graph unification

Definition

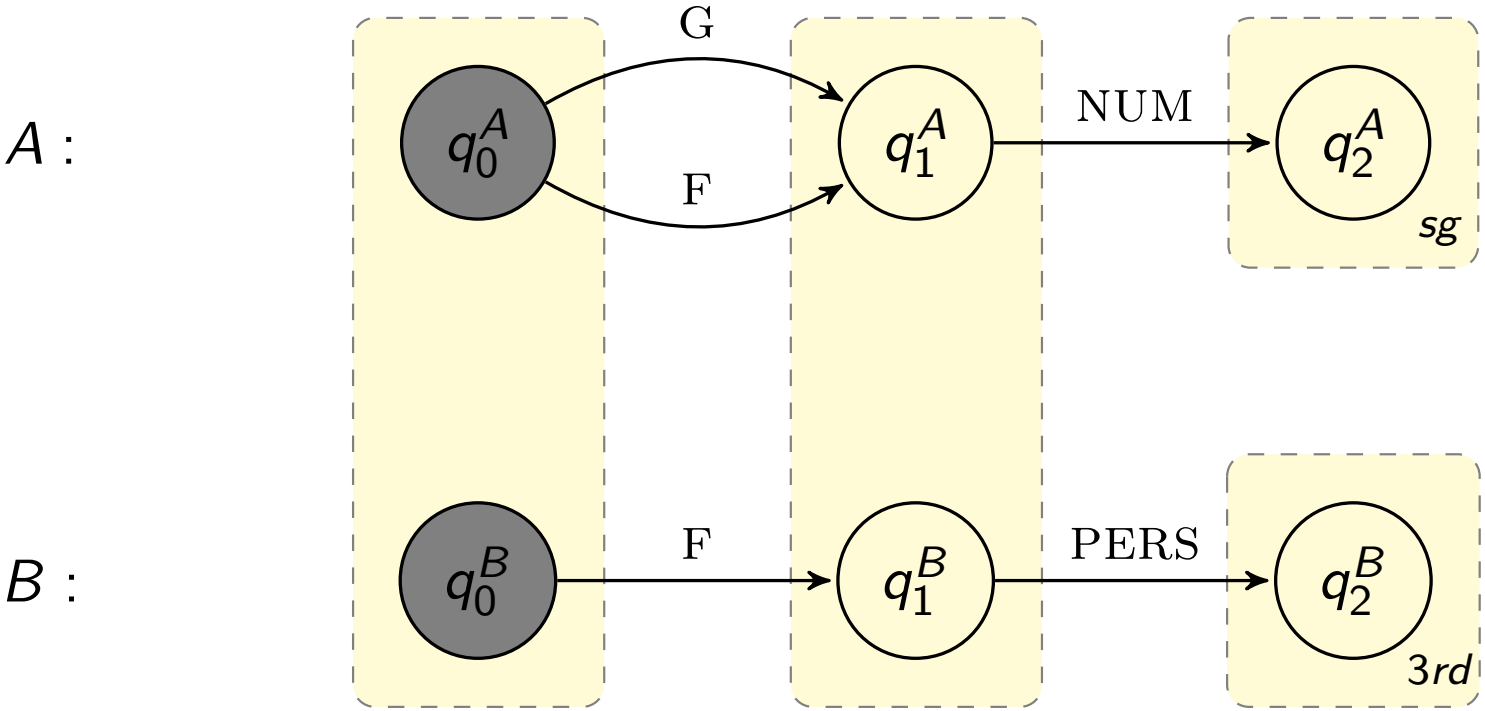
Let $A = \langle Q_A, \bar{q}_A, \delta_A, \theta_A \rangle$ and $B = \langle Q_B, \bar{q}_B, \delta_B, \theta_B \rangle$ with $Q_A \cap Q_B = \emptyset$ be two feature graphs. Let '$\overset{u}{\approx}$' be the least equivalence relation on $Q_A \cup Q_B$ such that:

- $\bar{q}_A \overset{u}{\approx} \bar{q}_B$
- for every $q_1, q_2 \in Q_A \cup Q_B$ and $f \in \textsc{Feats}$, if $q_1 \overset{u}{\approx} q_2, (\delta_A \cup \delta_B)(q_1, f)\downarrow$ and $(\delta_A \cup \delta_B)(q_2, f)\downarrow$, then $(\delta_A \cup \delta_B)(q_1, f) \overset{u}{\approx} (\delta_A \cup \delta_B)(q_2, f)$

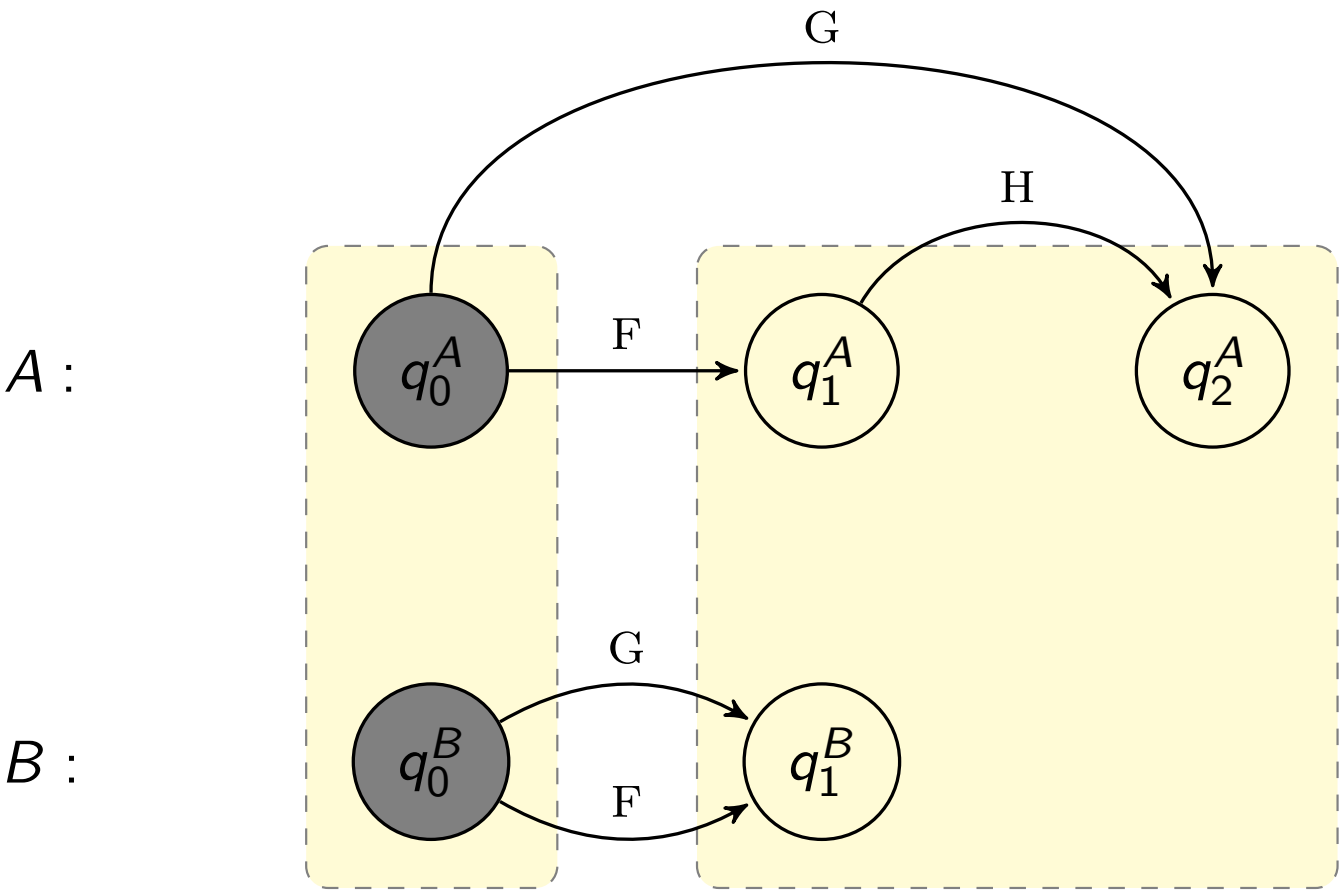
Feature graph unification

- The '$\overset{u}{\approx}$' relation partitions the nodes of $Q_A \cup Q_B$ to equivalence classes such that both roots are in the same class, and if some feature is defined for two nodes in one class, then the two nodes this feature leads to are also in one (possibly different) class.
- Clearly, the number of equivalence classes (called the *index* of $\overset{u}{\approx}$) is finite.
- The requirement that Q_A and Q_B be disjoint is essential here: we would want two nodes to be in the same equivalence class with respect to '$\overset{u}{\approx}$' only if they comply with the above definition; if we allowed a non-empty intersection of nodes, '$\overset{u}{\approx}$' could have been a different relation.

The $\overset{u}{\approx}$ relation



The $\overset{u}{\approx}$ relation



Type-respecting relation

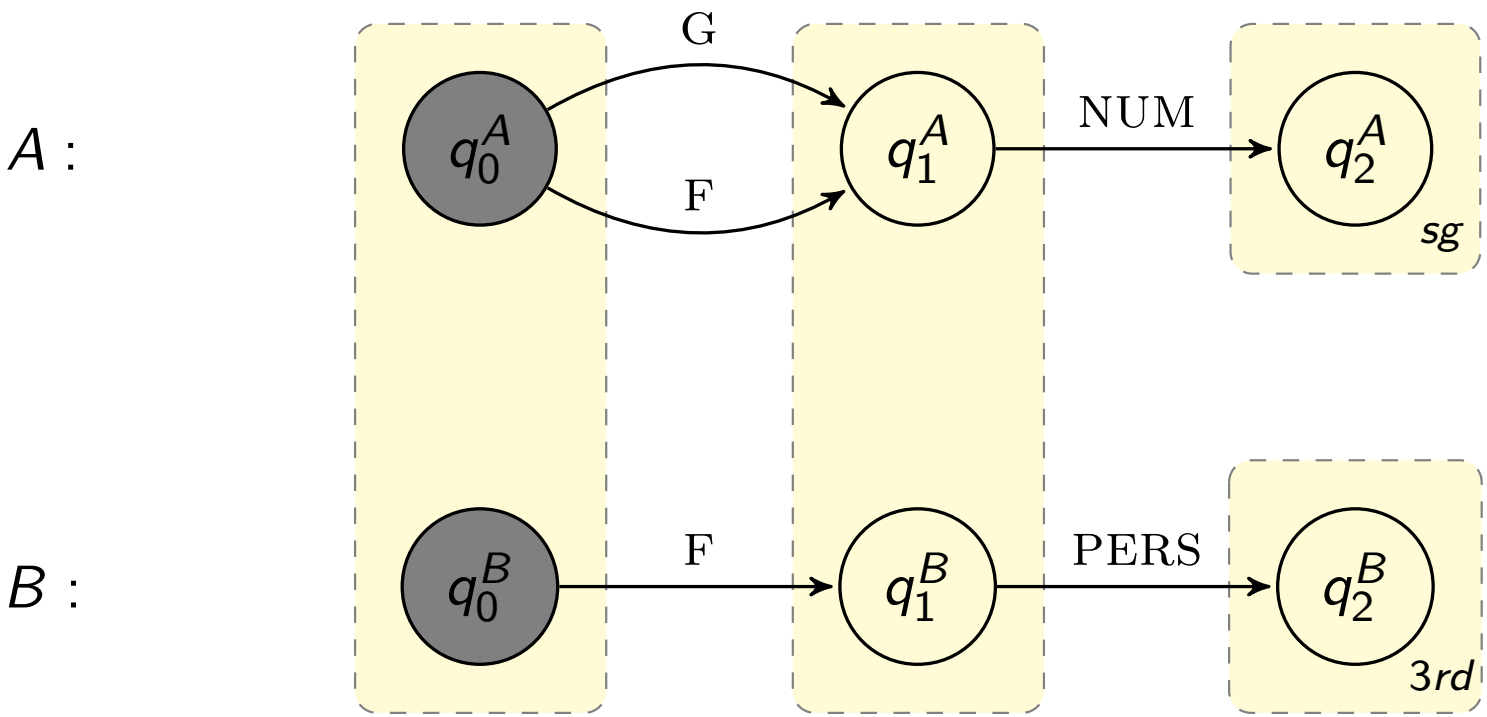
Definition

A binary relation '$\approx$' over the nodes of two feature structures $Q_A \cup Q_B$ is said to be **type respecting** iff for every node $q \in Q_A \cup Q_B$, if $(\theta_A \cup \theta_B)(q)\downarrow$ and $(\theta_A \cup \theta_B)(q) = a$, then for every node q' such that $q \approx q'$, q' is a sink and either $(\theta_A \cup \theta_B)(q')\uparrow$ or $(\theta_A \cup \theta_B)(q') = a$.

Type-respecting relation

- When is '$\overset{u}{\approx}$' *not* type respecting?
- The above condition can hold for a node $q \in Q_A \cup Q_B$ only if $(\theta_A \cup \theta_B)(q)\downarrow$; that is, q must be a sink in either A or B.
- The type respecting condition requires that all nodes that are equivalent to q be sinks, either unmarked or marked by the same atom.
- Since this is the only requirement, the relation is not type respecting if it maps two nodes, one of which is a marked sink and the other of which is either a non-sink or a sink with a different label, to the same equivalence class.
- A non-type respecting '$\overset{u}{\approx}$' is the only source for unification failure.

Type respecting $\overset{u}{\approx}$ relation



Feature graph unification

Lemma

If A and B have a common upper bound C, such that $A \sqsubseteq C$ through the morphism h_A and $B \sqsubseteq C$ through the morphism h_B, and if $q_A \in Q_A$ and $q_B \in Q_B$ are such that $q_A \overset{u}{\approx} q_B$, then $h_A(q_A) = h_B(q_B)$.

Feature graph unification

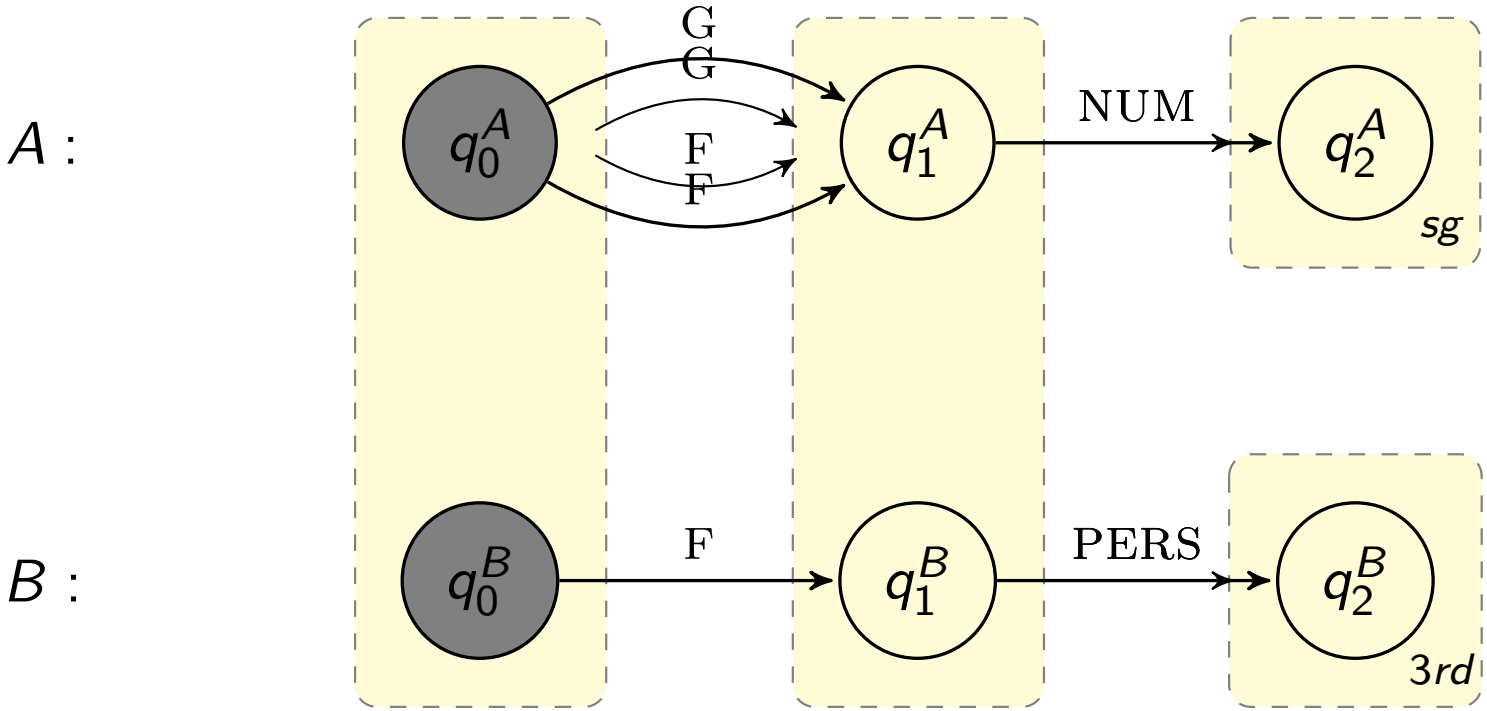
Definition (Feature graph unification)

Let A and B be two feature graphs such that Q_A and Q_B are disjoint. The **unification** of A and B, denoted $A \sqcup B$, is defined only if '$\overset{u}{\approx}$' is type respecting, in which case it is the feature graph $\langle Q, \bar{q}, \delta, \theta \rangle$, where:

- $Q = \{[q]_{\overset{u}{\approx}} \mid q \in (Q_A \cup Q_B)\}$
- $\bar{q} = [\bar{q}_1]_{\overset{u}{\approx}} (= [\bar{q}_2]_{\overset{u}{\approx}})$
- $\delta([q]_{\overset{u}{\approx}}, f) = \begin{cases} [q'']_{\overset{u}{\approx}} & \text{if there exists } q' \in [q]_{\overset{u}{\approx}} \text{ s.t. } (\delta_A \cup \delta_B)(q', f) = q'' \\ \text{undef.} & \text{if } (\delta_A \cup \delta_B)(q', f)\uparrow \text{ for all } q' \in [q]_{\overset{u}{\approx}} \end{cases}$
- $\theta([q]_{\overset{u}{\approx}}) = \begin{cases} (\theta_A \cup \theta_B)(q') & \text{if there exists } q' \in [q]_{\overset{u}{\approx}} \text{ s.t. } (\theta_A \cup \theta_B)(q')\downarrow \\ \text{undefined} & \text{if } (\theta_A \cup \theta_B)(q')\uparrow \text{ for all } q' \in [q]_{\overset{u}{\approx}} \end{cases}$

If $\overset{u}{\approx}$ is not type respecting, A and B are **inconsistent**.

Feature graph unification

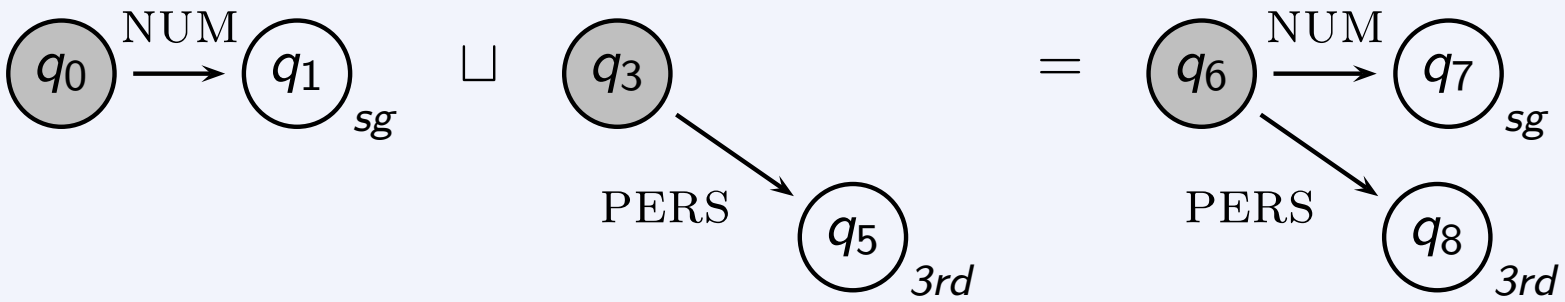


Unification

- To see that the result of unification is indeed a feature graph, observe that
  - $\langle Q, \bar{q}, \delta, \theta \rangle$ is connected because both A and B are connected;
  - it is finite since both A and B are (and hence the number of equivalence classes is finite);
  - and θ labels only sinks, since $\overset{u}{\approx}$ is type respecting.

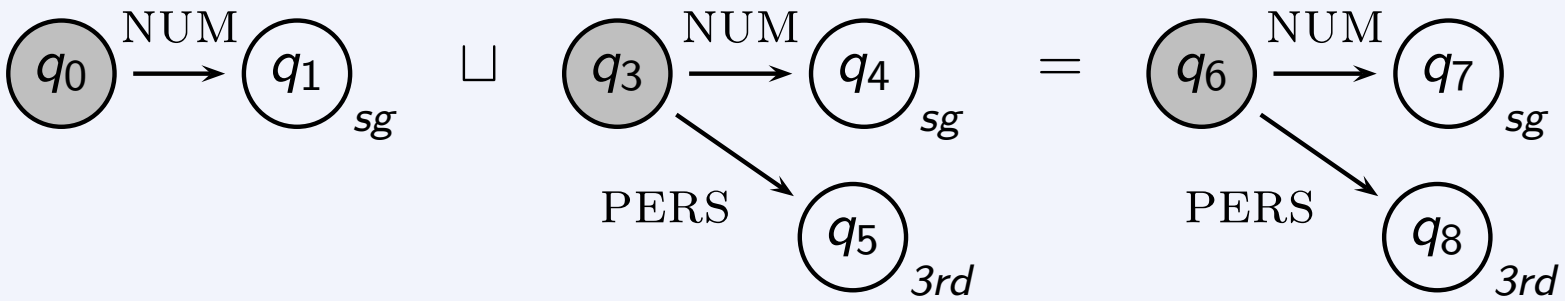
Unification

Example (Unification combines information)

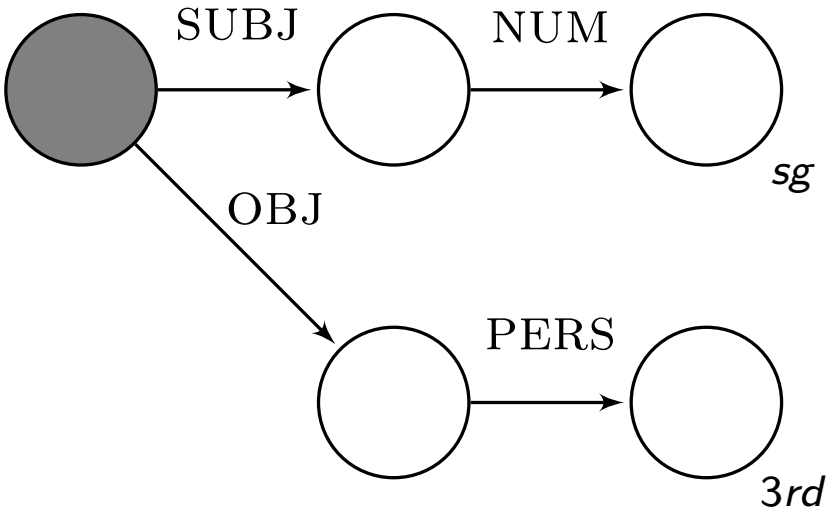


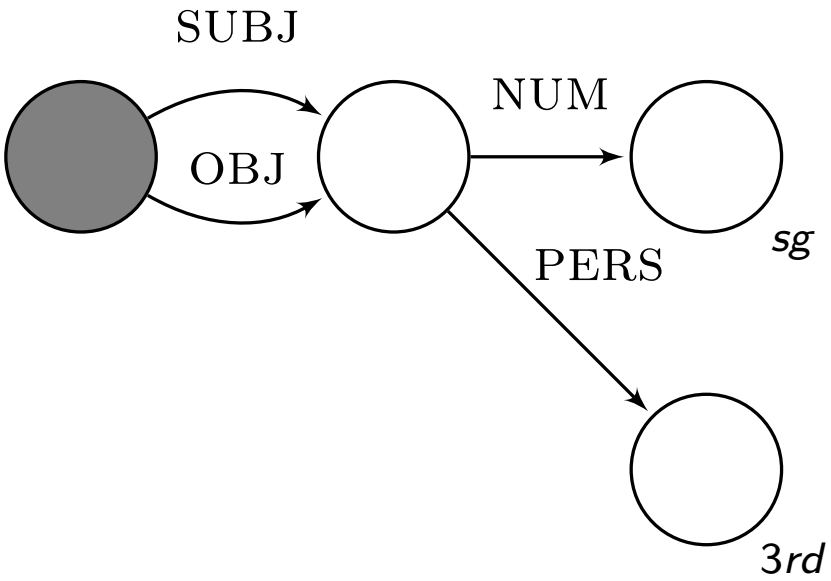
Unification

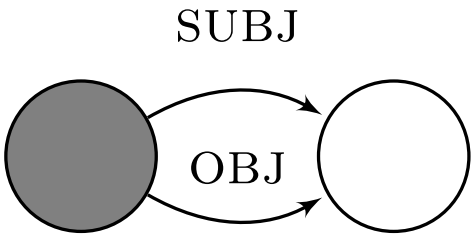
Example (Unification is absorbing)



Unification with reentrancies







Unification

Theorem

If A and B are inconsistent, they do not have a common upper bound. Otherwise, $C = A \sqcup B$ is a minimal upper bound of A and B with respect to (feature graph) subsumption.

Unification

- The previous theorem connects feature graph unification with feature structure unification.
- In order to compute $fs = fs_1 \hat{\sqcup} fs_2$, simply compute $A = A_1 \sqcup A_2$, where $A_1 \in fs_1$ and $A_2 \in fs_2$, and take $fs = [A]_\sim$.

Theorem

For all feature graphs A_1, A_2, if $A = A_1 \sqcup A_2$ then $[A]_\sim = [A_1]_\sim \hat{\sqcup} [A_2]_\sim$.

Generalization

- Unification is an information-combining operator: when two feature structures are compatible, their unification can be informally seen as a union of the information both structures encode.
- Sometimes, however, a dual operation is useful, analogous to the *intersection* of the information encoded in feature structures.
- This operation, which is much less frequently used in computational linguistics, is referred to as *anti-unification*, or *generalization*.

Generalization

- 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

Definition (Generalization)

The **generalization** (or **anti-unification**) of two feature structures fs_1 and fs_2, denoted $fs_1 \hat{\sqcap} fs_2$, is the greatest lower bound of fs_1 and fs_2.

Generalization

Example (Generalization)

- Generalization reduces information:

$$\begin{bmatrix}\textsc{num}: & sg\end{bmatrix} \hat{\sqcap} \begin{bmatrix}\textsc{pers}: & third\end{bmatrix} = [\,]$$

- Different atoms are inconsistent:

$$\begin{bmatrix}\textsc{num}: & sg\end{bmatrix} \hat{\sqcap} \begin{bmatrix}\textsc{num}: & pl\end{bmatrix} = \begin{bmatrix}\textsc{num}: & [\,]\end{bmatrix}$$

- Generalization is restricting:

$$\begin{bmatrix}\textsc{num}: & sg\end{bmatrix} \hat{\sqcap} \begin{bmatrix}\textsc{num}: & sg \\ \textsc{pers}: & third\end{bmatrix} = \begin{bmatrix}\textsc{num}: & sg\end{bmatrix}$$

Generalization

Example (Generalization)

- Empty feature structures are zero elements:

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

- Reentrancies can be lost:

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

Unification grammars

- Feature structures are the building blocks with which unification grammars are built, as they serve as the counterpart of the terminal and non-terminal symbols in CFGs.
- In order to define grammars and derivations, one needs some extension of feature structures to sequences thereof.
- *Multi-rooted feature structures* are aimed at capturing complex, ordered information and are used for representing rules and sentential forms of unification grammars.
  - Multi-rooted feature graphs, a natural extension of feature graphs
  - Multi-rooted feature structures, which are equivalence classes of isomorphic multi-rooted feature graphs
  - Multi-AVMs, which are an extension of AVMs, and show how they correspond to multi-rooted graphs.

Unification grammars

- Unification in context
- Forms and grammar rules
- Derivation
- Languages
- Derivation tress

Multi-rooted feature graphs

- We extend feature graphs to *multi-rooted feature graphs* (MRGs).
- Multi-rooted feature graphs are defined over the same signature (FEATS and ATOMS), which is assumed to be fixed

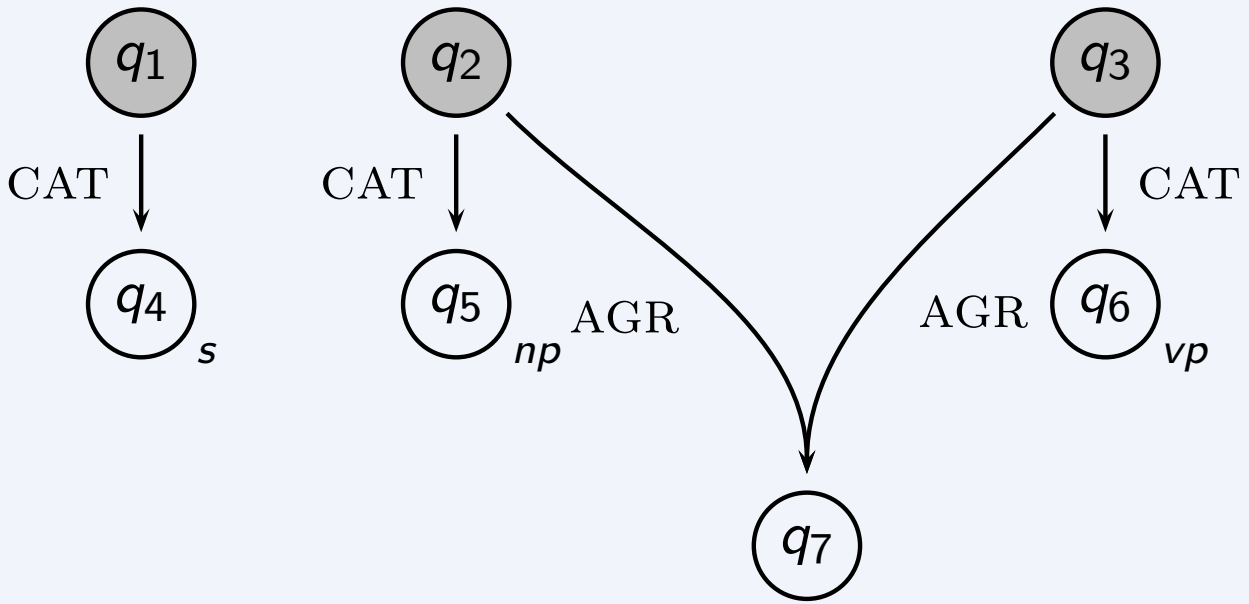
Definition (Multi-rooted feature graphs)

A **multi-rooted feature graph** (MRG) is a pair $\langle \bar{R}, G \rangle$ where $G = \langle Q, \delta, \theta \rangle$ is a finite, directed, labeled graph consisting of a non-empty, finite set Q of nodes (disjoint of FEATS and ATOMS), a partial function $\delta : Q \times \text{FEATS} \rightarrow Q$ specifying the arcs and a labeling function θ marking some of the sinks, and where $\bar{R}$ is an ordered list of distinguished nodes in Q called **roots**. G is not necessarily connected, but the union of all the nodes reachable from all the roots in $\bar{R}$ is required to yield exactly Q. The **length** of an MRG is the number of its roots, $|\bar{R}|$. λ denotes the empty MRG, where $Q = \emptyset$.

Multi-rooted feature graphs

Example (Multi-rooted feature graphs)

The following is an MRG, in which the shaded nodes (ordered from left to right) constitute the list of roots, $\bar{R}$



Multi-rooted feature graphs

- A multi-rooted feature graph is a directed, not necessarily connected, labeled graph with a designated sequence of nodes called *roots*
- It is a natural extension of feature graphs, the only difference being that the single root of a feature graph is extended here to a list in order to model the required structured information
- Meta-variables $\vec{A}$ range over MRGs, and Q, δ, θ and $\bar{R}$ – over their constituents
- We do not distinguish between an MRG of length 1 and a feature graph

Multi-rooted feature graphs

- Natural relations can be defined between MRGs and feature graphs
- First, note that if $\vec{A} = \langle \bar{R}, G \rangle$ is an MRG and $\bar{q}_i$ is a root in $\bar{R}$ then $\bar{q}_i$ naturally induces a feature graph $\vec{A}|_i = \langle Q_i, \bar{q}_i, \delta_i, \theta_i \rangle$, where:
  - Q_i is the set of nodes reachable from $\bar{q}_i$
  - $\delta_i = \delta|_{Q_i}$ (the restriction of δ to Q_i)
  - $\theta_i = \theta|_{Q_i}$ (the restriction of θ to Q_i).

Multi-rooted feature graphs

- One can view an MRG $\vec{A} = \langle \bar{R}, G \rangle$ as an ordered sequence $\langle A_1, \ldots, A_n \rangle$ of (not necessarily disjoint) feature graphs, where $A_i = \vec{A}|_i$ for $1 \leq i \leq n$
- Note that such an ordered list of feature structures is not a *sequence* in the mathematical sense:
  - removing a node accessible from one root can result in this node being removed from the graph accessible from some other root

Subgraphs

- Although MRGs are not element-disjoint sequences, it is possible to define substructures of them
- The roots of an MRG form a sequence of nodes
- Taking just a subsequence of the roots, and considering only the subgraph they induce (that is, the nodes that are accessible from these roots), a notion of *substructure* is naturally obtained

Subgraphs

Definition (Induced subgraphs)

The **subgraph** of a non-empty MRG $\vec{A} = \langle \bar{R}, G \rangle$, induced by j, k and denoted $\vec{A}^{j \cdots k}$, is defined only if $1 \leq i \leq j \leq n$, in which case it is the MRG $\langle \bar{R}', G' \rangle$ where $\bar{R}' = \langle q_j, \ldots, q_k \rangle$, $G' = \langle Q', \delta', \theta' \rangle$ and

- $Q' = \{q \mid \delta(\bar{q}, \pi) = q\}$ for some $\bar{q} \in \bar{R}'$ and some π
- $\delta'(q, f) = \delta(q, f)$ for every $q \in Q'$
- $\theta'(q) = \theta(q)$ for every $q \in Q'$

When the sequence is of length 1 we write $\vec{A}^i$ for $\vec{A}^{i \cdots i}$. As we identify a feature graph with an MRG of length 1, $\vec{A}^i = \vec{A}|_i$.

MRGs

- Since MRGs are a natural extension of feature graphs, many of concepts defined for the latter can be extended to the former
  - The transition function δ is extended from single features to *paths*
  - The set of paths of an MRG
  - The function *val*, associating a value with each path in a feature graph, is extended to MRGs.
  - Reentrancy and cyclicity
  - Isomorphism and subsumption

MRG paths

Definition (MRG paths)

The *paths* of a multi feature graph $\vec{A}$ are

$$\Pi(\vec{A}) = \{\langle i, \pi \rangle \mid \pi \in \textsc{Paths} \text{ and } \delta(\bar{q}_i, \pi)\downarrow\}$$

MRG path values

Definition (Path value)

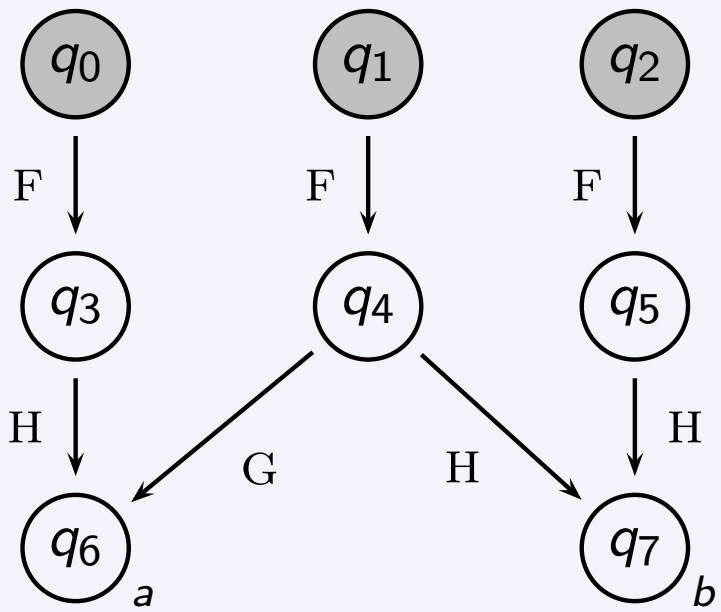
The **value** of a path $\langle i, \pi \rangle$ in an MRG $\vec{A}$, denoted by $val_{\vec{A}}(\langle i, \pi \rangle)$, is defined if and only if $\delta_{\vec{A}}(\bar{q}_i, \pi)\downarrow$, in which case it is the feature graph $val_{\vec{A}|_i}(\pi)$.

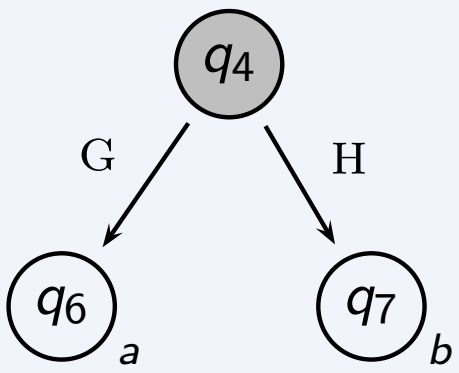
Note that the value of a path in an MRG is a (single-rooted) feature graph, not an MRG. In particular, $val_{\vec{A}}(\langle i, \pi \rangle)$ may include nodes which are roots in $\vec{A}$ but are not the root of the resulting feature graph. Clearly, an MRG may have two paths $\langle i_1, \pi_1 \rangle$ and $\langle i_2, \pi_2 \rangle$ where $\pi_1 = \pi_2$ even though $i_1 \neq i_2$.

MRG path values

Example (Path value)

$\vec{A}$, where $\bar{R} = \langle q_0, q_1, q_2 \rangle$ $\qquad$ $val_{\vec{A}}(\langle 2, \langle \mathrm{F} \rangle \rangle)$





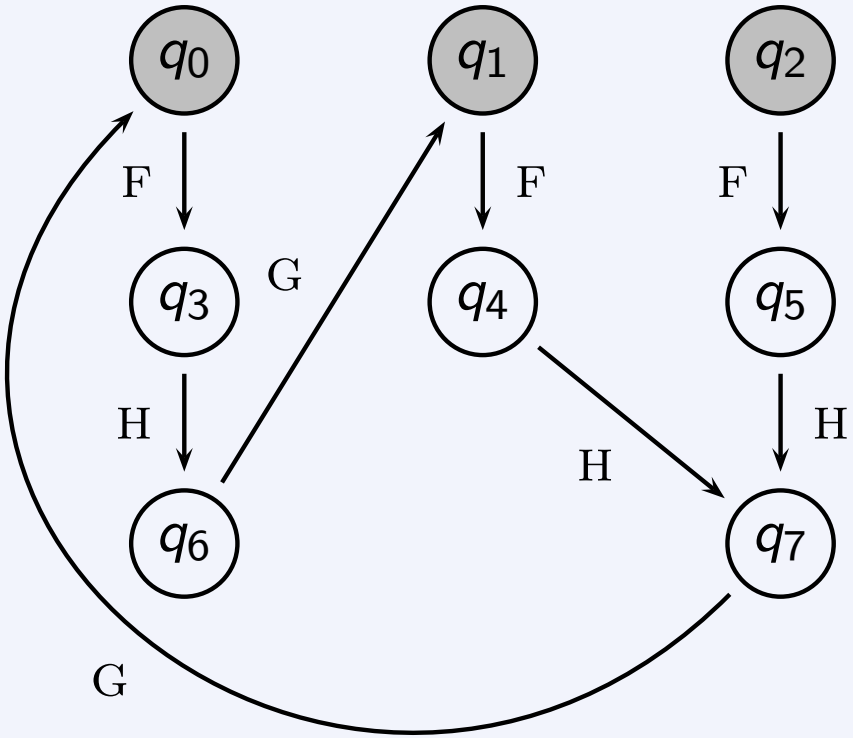
MRG reentrancy

- Two MRG paths are reentrant, denoted $\langle i, \pi_1 \rangle \overset{\vec{A}}{\leftrightsquigarrow} \langle j, \pi_2 \rangle$, if they share the same value: $\delta_{\vec{A}}(\bar{q}_i, \pi_1) = \delta_{\vec{A}}(\bar{q}_j, \pi_2)$
- A multi-rooted feature graph is *reentrant* if it has two distinct paths (possibly leaving different roots) that are reentrant
- An MRG $\vec{A}$ is *cyclic* if two paths $\langle i, \pi_1 \rangle, \langle i, \pi_2 \rangle \in \Pi(\vec{A})$, where π_1 is a proper subsequence of π_2, are reentrant: $\langle i, \pi_1 \rangle \overset{\vec{A}}{\leftrightsquigarrow} \langle i, \pi_2 \rangle$
- Here, the two paths must have the same index i, although they may "pass through" elements of $\vec{A}$ other than the i-th one

A cyclic MRG

Example (A cyclic MRG)

The following MRG $\vec{A} = \langle \bar{R}, G \rangle$, where $\bar{R} = \langle q_0, q_1, q_2 \rangle$, is cyclic:



Multi-rooted feature graph isomorphism

Definition (Multi-rooted feature graph isomorphism)

Two MRGs $\vec{A}_1 = \langle \bar{R}_1, G_1 \rangle$ and $\vec{A}_2 = \langle \bar{R}_2, G_2 \rangle$ are **isomorphic**, denoted $\vec{A}_1 \vec{\sim} \vec{A}_2$, iff they are of the same length, n, and there exists a one-to-one mapping $i : Q_1 \rightarrow Q_2$, called an **isomorphism**, such that:

- $i(\bar{q}_{1j}) = \bar{q}_{2j}$ for all $1 \leq j \leq n$;
- for all $q_1, q_2 \in Q_1$ and $f \in \textsc{Feats}$, $\delta_1(q_1, f) = q_2$ iff $\delta_2(i(q_1), f) = i(q_2)$; and
- for all $q \in Q_1$, $\theta_1(q) = \theta_2(i(q))$ (either both are undefined, or both are defined and equal).

Subsumption of multi-rooted feature graphs

Definition (Subsumption of multi-rooted feature graphs)

An MRG $\vec{A} = \langle \bar{R}, G \rangle$ **subsumes** an MRG $\vec{A}' = \langle \bar{R}', G' \rangle$, denoted $\vec{A} \vec{\sqsubseteq} \vec{A}'$, if $|\bar{R}| = |\bar{R}'|$ and there exists a total function $h : Q \rightarrow Q'$ such that:

- for every root $\bar{q}_i \in \bar{R}, h(\bar{q}_i) = \bar{q}'_i$
- for every $q \in Q$ and $f \in \textsc{Feats}$, if $\delta(q, f)\downarrow$ then $h(\delta(q, f)) = \delta'(h(q), f)$
- for every $q \in Q$, if $\theta(q)\downarrow$ then $\theta(q) = \theta'(h(q))$

The only difference from feature graph subsumption is that h is required to map each of the roots in $\bar{R}$ to its corresponding root in $\bar{R}'$. Notice that in order for two MRGs to be related by subsumption they must be of the same length.

Subsumption of multi-rooted feature graphs

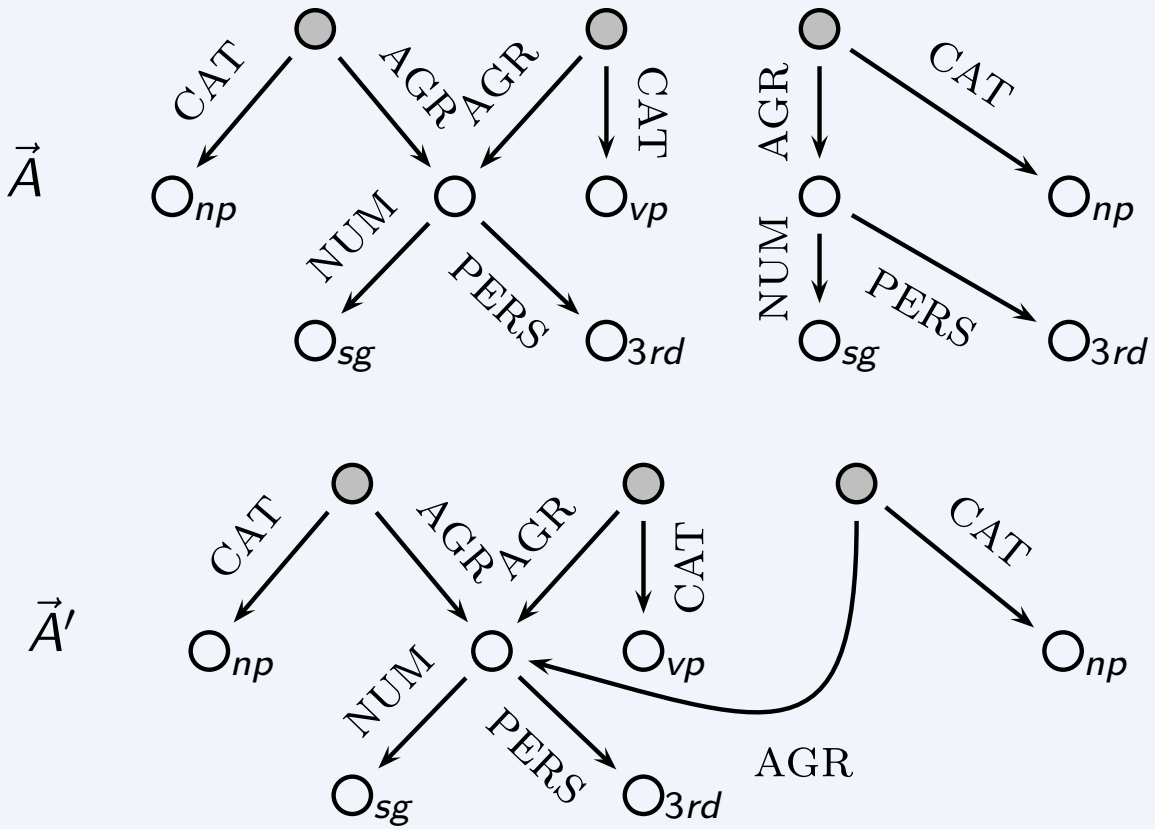
Example (MRG subsumption)

Feature graph subsumption can have three different effects: if $A \sqsubseteq B$, then B can have additional arcs, additional reentrancies or more marked atoms. The same holds for MRGs, with the observation that additional reentrancies can now occur among paths that originate at different roots:

Subsumption of multi-rooted feature graphs

Example (MRG subsumption)

Let $\vec{A}$ and $\vec{A}'$ be the following two MRGs. Then $\vec{A} \vec{\sqsubseteq} \vec{A}'$ but not $\vec{A}' \vec{\sqsubseteq} \vec{A}$.



Multi-AVMs

Definition

Given a signature $\mathcal{S}$, a **multi-AVM** (MAVM) of **length** $n \geq 0$ is a sequence $\langle M_1, \ldots, M_n \rangle$ such that for each i, $1 \leq i \leq n$, M_i is an AVM over the signature.

Multi-AVMs

- Meta-variables $\vec{M}$ range over multi-AVMs
- The **sub-AVMs** of $\vec{M}$ are $SubAVM(\vec{M}) = \bigcup_{1 \leq i \leq n} SubAVM(M_i)$
- Similarly to what we did for AVMs, we define the set of tags occurring in a multi-AVM $\vec{M}$ as $Tags(\vec{M})$
- Note that if $\vec{M} = \langle M_1, \ldots, M_n \rangle$ then $Tags(\vec{M}) = \bigcup_{1 \leq i \leq n} Tags(M_i)$ (where the union is not necessarily disjoint)
- Also, the set of sub-AVMs of $\vec{M}$ (including $\vec{M}$ itself) which are tagged by the same variable X is $TagSet(\vec{M}, X)$
- Here, too, $TagSet(\vec{M}, X) = \bigcup_{1 \leq i \leq n} TagSet(M_i, X)$
- We usually do not distinguish between a multi-AVM of length 1 and an AVM
- When depicting MAVMs graphically, we sometimes suppress the angular brackets which enclose the sequence of AVMs.

Multi-AVMs

Well-formedness and variable association are extended from AVMs to MAVMs in the natural way:

Definition (Well-formed MAVMs)

A multi-AVM $\vec{M}$ is well-formed iff for every variable $X \in Tags(\vec{M})$, $TagSet(\vec{M}, X)$ includes at most one non-empty AVM.

Definition (Variable association)

The **association** of a variable X in $\vec{M}$, denoted $assoc(\vec{M}, X)$, is the single non-empty AVM in $TagSet(\vec{M}, X)$; if all the members of $TagSet(\vec{M}, X)$ are empty, then $assoc(\vec{M}, X) = X\,[\,]$.

Multi-AVMs

Example (Multi-AVMs)

Consider the following multi-AVM $\vec{M}$, whose length is 3:

$$\left\langle \boxed{2}\begin{bmatrix}\text{F}: & \boxed{9}\begin{bmatrix}\text{H}: & \boxed{1}[\,]\end{bmatrix}\end{bmatrix}, \boxed{1}\begin{bmatrix}\text{F}: & \boxed{8}\begin{bmatrix}\text{G}: & \boxed{7}a \\ \text{H}: & \boxed{2}[\,]\end{bmatrix}\end{bmatrix}, \boxed{6}\begin{bmatrix}\text{F}: & \boxed{5}\begin{bmatrix}\text{H}: & \boxed{2}[\,]\end{bmatrix}\end{bmatrix}\right\rangle$$

$Tags(\vec{M}) = \{\boxed{1}, \boxed{2}, \boxed{5}, \boxed{6}, \boxed{7}, \boxed{8}, \boxed{9}\}$. $\vec{M}$ is well-formed:

$$TagSet(\vec{M}, \boxed{1}) = \left\{\boxed{1}[\,], \boxed{1}\begin{bmatrix}\text{F}: & \boxed{8}\begin{bmatrix}\text{G}: & \boxed{7}a \\ \text{H}: & \boxed{2}[\,]\end{bmatrix}\end{bmatrix}\right\}$$

$$TagSet(\vec{M}, \boxed{2}) = \left\{\boxed{2}[\,], \boxed{2}\begin{bmatrix}\text{F}: & \boxed{9}\begin{bmatrix}\text{H}: & \boxed{1}[\,]\end{bmatrix}\end{bmatrix}\right\}$$

Therefore,

$$assoc(\vec{M}, \boxed{1}) = \boxed{1}\begin{bmatrix}\text{F}: & \boxed{8}\begin{bmatrix}\text{G}: & \boxed{7}a \\ \text{H}: & \boxed{2}[\,]\end{bmatrix}\end{bmatrix}, assoc(\vec{M}, \boxed{2}) = \boxed{2}\begin{bmatrix}\text{F}: & \boxed{9}\begin{bmatrix}\text{H}: & \boxed{1}[\,]\end{bmatrix}\end{bmatrix}$$

Multi-AVMs

- The same variable can tag different sub-AVMs of different elements in the sequence
- In other words, the *scope* of variables is extended from single AVMs to multi-AVMs
- This leads to an interpretation of variables (in multi-AVMs) which hampers the view of multi-AVMs as sequences of AVMs
- Recall that we interpret multiple occurrence of the same variable within a single AVM as denoting value sharing; hence the definition of well-formed AVMs, and the convention that when a variable occurs more than once in an AVM, its association can be stipulated next to any of its occurrences
- As in the other views, when multi-AVMs are concerned, this convention implies that removing an element from a multi-AVM can affect other elements, in contradiction to the usual concept of sequences

MAVM subsumption

Definition (Multi-AVM subsumption)

Let $\vec{M}$, $\vec{M}'$ be two MAVMs of the same length n and over the same signature. $\vec{M}$ **subsumes** $\vec{M}'$, denoted $\vec{M} \vec{\preceq} \vec{M}'$, if the following conditions hold:

1. for all i, $1 \leq i \leq n$, $M_i \preceq M'_i$;
2. if $\langle i, \pi_1 \rangle \overset{\vec{M}}{\leftrightsquigarrow} \langle j, \pi_2 \rangle$ then $\langle i, \pi_1 \rangle \overset{\vec{M}'}{\leftrightsquigarrow} \langle j, \pi_2 \rangle$.

MAVM subsumption

Example (MAVM subsumption)

Let $\vec{M}$ and $\vec{M}'$ be the following two MAVMs (of length 3):

$$\vec{M}: \quad \boxed{1}\begin{bmatrix}\text{CAT}: & np \\ \text{AGR}: & \boxed{4}\end{bmatrix} \boxed{2}\begin{bmatrix}\text{CAT}: & vp \\ \text{AGR}: & \boxed{4}\begin{bmatrix}\text{NUM}: & sg \\ \text{PERS}: & 3rd\end{bmatrix}\end{bmatrix} \boxed{3}\begin{bmatrix}\text{CAT}: & np \\ \text{AGR}: & \boxed{6}\begin{bmatrix}\text{NUM}: & sg \\ \text{PERS}: & 3rd\end{bmatrix}\end{bmatrix}$$

$$\vec{M}': \quad \boxed{1}\begin{bmatrix}\text{CAT}: & np \\ \text{AGR}: & \boxed{4}\end{bmatrix} \boxed{2}\begin{bmatrix}\text{CAT}: & vp \\ \text{AGR}: & \boxed{4}\begin{bmatrix}\text{NUM}: & sg \\ \text{PERS}: & 3rd\end{bmatrix}\end{bmatrix} \boxed{3}\begin{bmatrix}\text{CAT}: & np \\ \text{AGR}: & \boxed{4}\end{bmatrix}$$

Then $\vec{M} \preceq \vec{M}'$ but not $\vec{M}' \preceq \vec{M}$.

MAVM subsumption

- The second clause of the definition may seem redundant: if for all i, $1 \leq i \leq n$, $M_i \preceq M'_i$, then in particular all the reentrancies of M_i are all reentrancies in M'_i; why then is the second clause necessary?
- The answer lies in the possibility of reentrancies *across* elements in multi-AVMs
- Such reentrancies are a "global" property of multi-AVMs, which is not reflected in any of the elements in isolation

MAVM Renaming

Definition (Renaming)

Let $\vec{M}_1$ and $\vec{M}_2$ be two MAVMs. $\vec{M}_2$ is a **renaming** of $\vec{M}_1$, denoted $\vec{M}_1 \vec{\simeq} \vec{M}_2$, iff $\vec{M}_1 \vec{\preceq} \vec{M}_2$ and $\vec{M}_2 \vec{\preceq} \vec{M}_1$.

Multi-AVM to MRG mapping

Definition (Multi-AVM to MRG mapping)

Let $\vec{M} = \langle M_1, \ldots, M_n \rangle$ be a well-formed multi-AVM of length n. The MRG image of $\vec{M}$ is $\varphi(\vec{M}) = \langle \bar{R}, G \rangle$, with $\bar{R} = \langle \bar{q}_1, \ldots, \bar{q}_n \rangle$ and $G = \langle Q, \delta, \theta \rangle$, where:

- $Q = Tags(\vec{M})$
- $\bar{q}_i = tag(M_i)$ for $1 \leq i \leq n$
- for all $X \in Tags(\vec{M})$ and $f \in \textsc{Feats}$, $\delta(X, f) = Y$ if $\langle X, f, Y \rangle \in \textsc{Arcs}(\vec{M})$, and
- for all $X \in Tags(\vec{M})$ and $a \in \textsc{Atoms}$, $\theta(X) = a$ if $assoc(\vec{M}, X)$ is the atomic AVM $X(a)$, and is undefined otherwise.

Multi-AVM to MRG mapping

Example (Multi-AVM to multi-rooted feature graph mapping)

Consider the following multi-AVM $\vec{M}$:

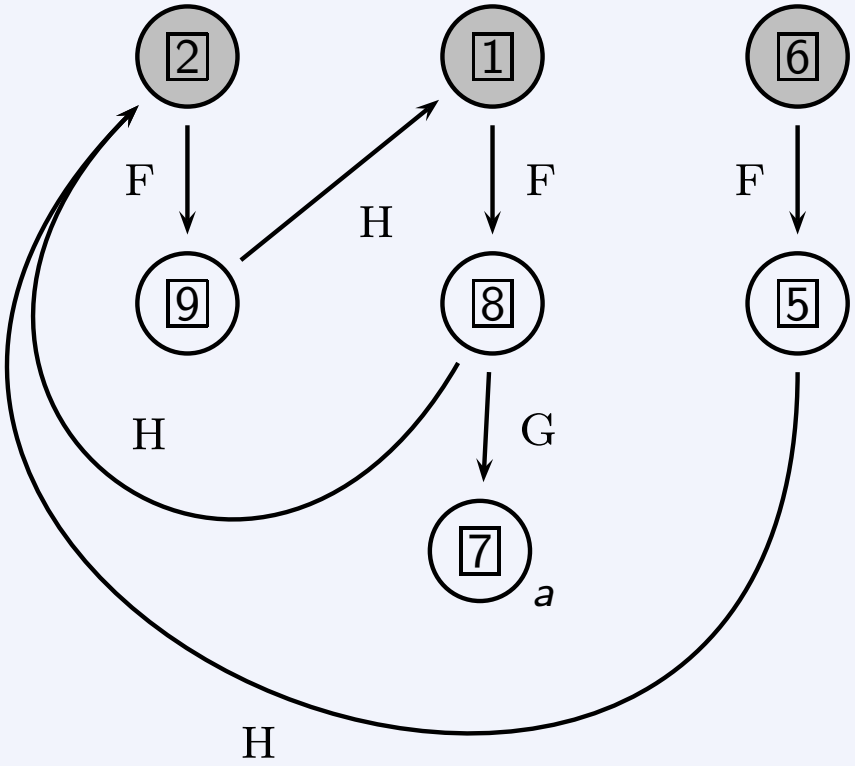
$$\boxed{2}\begin{bmatrix}\text{F}: & \boxed{9}\begin{bmatrix}\text{H}: & \boxed{1}[\,]\end{bmatrix}\end{bmatrix} \quad \boxed{1}\begin{bmatrix}\text{F}: & \boxed{8}\begin{bmatrix}\text{G}: & \boxed{7}a \\ \text{H}: & \boxed{2}[\,]\end{bmatrix}\end{bmatrix} \quad \boxed{6}\begin{bmatrix}\text{F}: & \boxed{5}\begin{bmatrix}\text{H}: & \boxed{2}[\,]\end{bmatrix}\end{bmatrix}$$

Observe that it is well-formed, as the variables that occur more than once ($\boxed{1}$ and $\boxed{2}$) have only one non-empty occurrence each. The set of variables of $\vec{M}$ is $Tags(\vec{M}) = \{\boxed{1}, \boxed{2}, \boxed{5}, \boxed{6}, \boxed{7}, \boxed{8}, \boxed{9}\}$, which will also be the set of nodes Q in $\varphi(\vec{M})$. The sequence of roots $\bar{R}$ is the sequence of variables tagging the AVM elements of $\vec{M}$, namely $\langle \boxed{2}, \boxed{1}, \boxed{6} \rangle$.

Multi-AVM to MRG mapping

Example (Multi-AVM to multi-rooted feature graph mapping)

The obtained graph is:



Multi-AVM to MRG mapping

Proposition

Let $\vec{M}_1, \vec{M}_2$ *be two multi-AVMs. Then:*

- $\Pi(\vec{M}) = \Pi(\varphi(\vec{M}))$
- $\langle i, \pi_1 \rangle \overset{\vec{M}}{\leftrightsquigarrow} \langle j, \pi_2 \rangle$ *iff* $\langle i, \pi_1 \rangle \overset{\varphi(\vec{M})}{\leftrightsquigarrow} \langle j, \pi_2 \rangle$
- $\vec{M}_1 \vec{\preceq} \vec{M}_2$ *iff* $\varphi(\vec{M}_1) \vec{\sqsubseteq} \varphi(\vec{M}_2)$

Unification revisited

- We defined the unification operation for feature structures
- We now extend the definition to multi-rooted structures; we define two variants of the operation:
  - one which unifies two same-length structures and produces their least upper bound with respect to subsumption
  - *unification in context*, which combines the information in two feature structures, each of which may be an element in a larger structure

Two AMRS unification operations

Example (Two AMRS unification operations)

$$\begin{array}{cccccc} [\,] & [\,] & [\,] & \cdots & & [\,] \\ [\,] & [\,] & [\,] & \cdots & & [\,] \\ \hline & & & & & \\ [\,] & [\,] & [\,] & \cdots & & [\,] \end{array}$$

Same-length AMRS unification

$$\begin{array}{ccccc} & & [\,] & & \\ & & [\,] & & \\ [\,] & [\,] & [\,] & [\,] & [\,] \\ & & [\,] & & \end{array}$$

Unification in context

MRS unification

Example (MRS unification)

Let

$$\sigma = \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & \boxed{4} \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{4} \end{bmatrix}$$

$$\rho = \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & pl \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & [\,] \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & pl \end{bmatrix}$$

Then

$$\sigma \sqcup \rho = \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & \boxed{4}pl \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{4} \end{bmatrix}$$

Unification in context

Example (Unification in context)

Let

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

Unifying the first element in σ with the first element in ρ in the contexts of σ and ρ, we obtain $(\sigma, 1) \sqcup (\rho, 1) = (\sigma', \rho')$:

$$\sigma' = \begin{bmatrix} \text{F}: & \boxed{1}a \\ \text{G}: & \boxed{2}b \end{bmatrix} \begin{bmatrix} \text{H}: & \boxed{2} \end{bmatrix}, \quad \rho' = \begin{bmatrix} \text{F}: & \boxed{3}a \\ \text{G}: & \boxed{4}b \end{bmatrix} \begin{bmatrix} \text{H}: & \boxed{3} \end{bmatrix}$$

Note that both operands of the unification are modified.

Unification in context

Theorem

If $\langle \sigma', \rho' \rangle = (\sigma, i) \sqcup (\rho, j)$ *then* $\sigma'^i = \rho'^j = \sigma^i \sqcup \rho^j$.

Unification in context

Theorem

Let σ, ρ be two AMRSs and i, j be indexes such that $i \leq len(\sigma)$ and $j \leq len(\rho)$. Then $\langle \sigma', \rho' \rangle = (\sigma, i) \sqcup (\rho, j)$ iff

$\sigma' = \min_{\hat{\sqsubseteq}} \{\sigma'' \mid |\sigma \overset{\hat{\rightarrow}}{\sqsubseteq} \sigma''$ *and* $\rho^j \hat{\preceq} \sigma''^i\}$ *and*

$\rho' = \min_{\hat{\sqsubseteq}} \{\rho'' \mid \rho \overset{\hat{\rightarrow}}{\sqsubseteq} \rho''$ *and* $\sigma^i \hat{\preceq} \rho''^j\}$.

Rules and grammars

- Like context free grammars, unification grammars are defined over an alphabet
- As the grammars that are of most interest to us are of natural languages, and since sentences in natural languages are not just strings of symbols, but rather strings of *words*, we add to the signature an *alphabet*, a fixed set WORDS of words (in addition to the fixed sets FEATS and ATOMS)
- Meta-variables w_i, w_j etc. are used to refer to elements of WORDS, w to refer to strings over WORDS.

Rules and grammars

- We also adopt here the distinction between *phrasal* and *terminal* rules
- The former cannot have elements of WORDS in their bodies; the latter have only a single word as their body
- We refer to the collection of terminal rules as the *lexicon*: it associates with terminals, members of WORDS, (abstract) feature structures that are their *categories*
- For every word $w_i \in$ WORDS the lexicon specifies a finite set of abstract feature structures $\mathcal{L}(w_i)$
- If $\mathcal{L}(w_i)$ is a singleton then w_i is unambiguous, and if it is empty then w_i is not a member of the language defined by the lexicon.

Lexicon

Definition (Lexicon)

Given a signature of features $\textsc{Feats}$ and atoms $\textsc{Atoms}$, and a set $\textsc{Words}$ of terminal symbols, a **lexicon** is a finite-range function $\mathcal{L} : \textsc{Words} \rightarrow 2^{\mathcal{AFS}(\textsc{Feats},\textsc{Atoms})}$.

Lexicon

Example (Lexicon)

Following is a lexicon $\mathcal{L}$ over a signature consisting of $\textsc{Feats} = \{\textsc{cat},\textsc{num},\textsc{case}\}$, $\textsc{Atoms} = \{d, n, v, sg, pl\}$, and $\textsc{Words} = \{\text{two, sheep, sleep}\}$:

$$\mathcal{L}(\text{two}) = \left\{ \begin{bmatrix} \textsc{cat}: & d \\ \textsc{num}: & pl \end{bmatrix} \right\} \quad \mathcal{L}(\text{sheep}) = \left\{ \begin{bmatrix} \textsc{cat}: & n \\ \textsc{num}: & [\,] \\ \textsc{case}: & [\,] \end{bmatrix} \right\}$$

$$\mathcal{L}(\text{sleep}) = \left\{ \begin{bmatrix} \textsc{cat}: & v \\ \textsc{num}: & pl \end{bmatrix} \right\}$$

Lexicon

Example (Lexicon)

An an alternative to the previous lexical entry of sheep above, the grammar writer may prefer the following lexical entry:

$$\mathcal{L}(\text{sheep}) = \left\{ \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & sg \\ \text{CASE}: & [\,] \end{bmatrix}, \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & [\,] \end{bmatrix} \right\}$$

Lexicon

Example (Lexicon, rule-format)

To depict the lexicon specification above, we usually use the following notation:

$$\text{sheep} \longrightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & sg \\ \text{CASE}: & [\,] \end{bmatrix} \qquad \text{sheep} \longrightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & [\,] \end{bmatrix}$$

Lexicon

- When a string of words w is given, it is possible to construct an AMRS σ_w for the lexical entries of the words in w, such that no two elements of σ_w share paths
- Such an AMRS is simply the concatenation of the lexical entries of the words in w
- In general, there may be several such AMRSs, as each word in w can have multiple elements in its category
- The set of such AMRSs is the *pre-terminals* of w

Pre-terminals

Definition (Pre-terminals)

Let $w = w_1 \dots w_n \in \textsc{Words}^+$. $PT_w(j,k)$ is defined iff $1 \leq j,k \leq n$, in which case it is the set of AMRSs $\{\langle A_j \cdot A_{j+1} \cdots A_k \rangle \mid A_i \in \mathcal{L}(w_i)$ for $j \leq i \leq k\}$. If $j > k$ (i.e., $w = \epsilon$), then $PT_w(j,k) = \{\lambda\}$. The subscript w is omitted when it is clear from the context.

Pre-terminals

Example (Pre-terminals)

Consider the string of words $w =$ two sheep sleep and the lexicon of the previous example. There is exactly one element in $PT_w(1,3)$; this is the AMRS

$$\begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & pl \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & [\,] \\ \text{CASE}: & [\,] \end{bmatrix} \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & pl \end{bmatrix}$$

Notice that there is no sharing of variables among different feature structures in this AMRS. As AMRSs are depicted using multi-AVMs here, the variables in the above multi-AVM are *chosen* such that unintended reentrancies are avoided.

Pre-terminals

Example (Pre-terminals)

Now assume that the word sheep is represented as an ambiguous word: its category contains two feature structures, namely

$$\mathcal{L}(\text{sheep}) = \left\{ \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & sg \\ \text{CASE}: & [\,] \end{bmatrix}, \quad \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & [\,] \end{bmatrix} \right\}$$

Then $PT_w(1,3)$ has two members:

$$\left\{ \begin{array}{l} \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & pl \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & sg \\ \text{CASE}: & [\,] \end{bmatrix} \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & pl \end{bmatrix}, \\ \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & pl \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & [\,] \end{bmatrix} \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & pl \end{bmatrix} \end{array} \right\}$$

Rules

Definition (Rules)

A (phrasal) **rule** is an AMRS of length $n > 0$ with a distinguished first element. If σ is a rule then σ^1 is its **head** and $\sigma^{2..n}$ is its **body**. We adopt a convention of depicting rules with an arrow ($\rightarrow$) separating the head from the body.

- Since a rule is simply an AMRS, there can be reentrancies among its elements: both between the head and (some element of) the body and among elements in its body.
- Notice that the definition supports ϵ-rules, i.e., rules with null bodies

Rules

Example (Rules as AMRSs)

As every AMRS can be interpreted as a rule, so can the following:

$$\begin{bmatrix}\text{CAT}: & s\end{bmatrix} \quad \rightarrow \quad \begin{bmatrix}\text{CAT}: & np \\ \text{AGR}: & \boxed{4}\end{bmatrix} \qquad \begin{bmatrix}\text{CAT}: & v \\ \text{AGR}: & \boxed{4}\end{bmatrix}$$

Rules

Example (Rules as AMRSs)

Rules can also propagate information between the mother and any of the daughters using reentrancies between paths originating in the head of the rule and paths originating from one of the body elements, as below.

$$\begin{bmatrix} \text{CAT}: & s \\ \text{SUBJ}: & \boxed{1} \end{bmatrix} \rightarrow \boxed{1}\begin{bmatrix} \text{CAT}: & np \\ \text{AGR}: & \boxed{2} \end{bmatrix} \quad \begin{bmatrix} \text{CAT}: & v \\ \text{AGR}: & \boxed{2} \end{bmatrix}$$

Rules

- The rules of the example employ feature structures that include the feature CAT, encoding the major part-of-speech category of phrases
- While this is useful and natural, it is by no means obligatory
- Unification rules can encode such information in other ways (e.g., via a different feature, or as a collection of features); or they may not encode it at all
- In the general case, a unification rule is not required to have a *context-free skeleton*, a feature whose values constitute a context-free backbone that drives the derivation
- Some unification-based grammar theories do indeed maintain a context-free skeleton (LFG is a notable example), while others (like HPSG) do not

Rules

- We introduce a shorthand notation in the presentation of grammars:
  - When two rules have the same head, we list the head only once and separate the bodies of the different rules with '|' (following the convention of context-free grammars)
- Note, however, that the scope of variables is still limited to a single rule, so that multiple occurrences of the same variable within the bodies of two different rules are unrelated
- Additionally, we may use the same variable (e.g., $\boxed{4}$) in several rules
- It should be clear by now that these multiple uses are unrelated to each other, as the scope of variables is limited to a single rule

Unification grammars

Definition (Unification grammars)

A **unification grammar (UG)** $G = (\mathcal{L}, \mathcal{R}, A_s)$ over a signature $\textsc{Atoms}$ of atoms and $\textsc{Feats}$ of features consists of a lexicon $\mathcal{L}$, a finite set of rules $\mathcal{R}$ and a **start symbol** A_s that is an abstract feature structure.

Unification grammars

Example (G_u, a unification grammar)

$$\begin{bmatrix}\text{CAT}: & s\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & nom\end{bmatrix} \begin{bmatrix}\text{CAT}: & v \\ \text{NUM}: & \boxed{4}\end{bmatrix}$$

$$\begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & d \\ \text{NUM}: & \boxed{4}\end{bmatrix} \begin{bmatrix}\text{CAT}: & n \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix}$$

$$\begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & pron \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix}$$

Unification grammars

Example (G_u, a unification grammar)

$$\text{sleep} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & pl \end{bmatrix} \qquad \text{sleeps} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & sg \end{bmatrix}$$

$$\text{lamb} \rightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & sg \\ \text{CASE}: & [\,] \end{bmatrix} \qquad \text{lambs} \rightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & [\,] \end{bmatrix}$$

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

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

Derivations

- The *language* generated by UGs is defined in a parallel way to the definition of languages generated by context-free grammars:
- first, we define *derivations*, analogously to the context-free derivations
- The reflexive transitive closure of the derivation relation is the basis for the definition of languages
- For the following discussion fix a particular grammar $G = (\mathcal{L}, \mathcal{R}, A_s)$

Derivations

- Derivation is a relation that holds between two *forms*, σ_1 and σ_2, each of which is an AMRS
- To define it formally, two concepts have to be taken care of:
  - An element of σ_1 has to be *matched* against the head of some grammar rule, ρ
  - The body of ρ must *replace* the selected element in σ_1, thus producing σ_2
- Matching involves unification, and unification must be computed *in context*: that is, when the selected element of σ_1 is unified with the head of ρ, other elements in σ_1 or in ρ may be affected due to reentrancy
- This possibility must be taken care of when *replacing* the selected element with the body of ρ

Derivations

Definition (Derivation)

An AMRS σ_1 of length k **derives** an AMRS σ_2 (denoted $\sigma_1 \Rightarrow \sigma_2$) iff for some $j \leq k$ and some rule $\rho \in \mathcal{R}$ of length n,

- $(\sigma_1, j) \sqcup (\rho, 1) = (\sigma_1', \rho')$, and
- σ_2 is the replacement of the j-th element of σ_1 with the body of ρ (details suppressed)

The reflexive transitive closure of '$\Rightarrow$' is '$\overset{*}{\Rightarrow}$'. We write $\sigma \overset{l}{\Rightarrow} \rho$ when σ derives ρ in l steps.

Derivation step

Example (Derivation step)

Suppose that

$$\sigma_1 = \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{1} \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \end{bmatrix}$$

is a (sentential) form and that

$$\rho = \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{2} \\ \text{CASE}: & \boxed{3} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & \boxed{2} \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & \boxed{2} \\ \text{CASE}: & \boxed{3} \end{bmatrix}$$

is a rule. Assume further that the selected element j in σ_1 is the first one. Applying the rule ρ to the form σ_1, it is possible to construct a derivation $\sigma_1 \Rightarrow \sigma_2$.

Derivation step

Example (Derivation step)

First, compute $(\sigma_1, 1) \sqcup (\rho, 1) = (\sigma'_1, \rho')$:

$$\sigma'_1 = \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{1} \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \end{bmatrix}$$

$$\rho' = \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{2} \\ \text{CASE}: & \boxed{3} nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & \boxed{2} \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & \boxed{2} \\ \text{CASE}: & \boxed{3} \end{bmatrix}$$

Derivation step

Example (Derivation step)

Now, the first element of σ_1' is replaced by the body of ρ'. This operation results in a new AMRS, σ_2, of length 3: the first two elements are the body of ρ', and the last element is the remainder of σ_1', after its first element has been eliminated; that is, the last element of σ_1'. A simple replacement would have resulted in the following AMRS:

$$\begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & \boxed{2} \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & \boxed{2} \\ \text{CASE}: & \boxed{3}nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \end{bmatrix}$$

Obviously, this is not the expected result!

Derivation step

Example (Derivation step)

Since the path $(1, \text{NUM})$ in σ_1 is reentrant with $(2, \text{NUM})$ (indicated by the tag $\boxed{1}$), and since the path $(1, \text{NUM})$ in the rule ρ is reentrant with the paths $(2, \text{NUM})$ and $(3, \text{NUM})$ (the tag $\boxed{3}$), one would expect that the sharing between the NUM values of the noun phrase and the verb phrase in σ_1 would manifest itself as a sharing between this feature's values of the determiner, the noun and the verb phrase in σ_2.

This is what the last clause in the definition of derivation guarantees. The result is:

$$\sigma_2 = \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & \boxed{4} \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{5}nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{4} \end{bmatrix}$$

Derivation

Example (Derivation)

Consider the grammar G_u. A derivation with G_u can start with a form of length 1, consisting of

$$\sigma_1 = \begin{bmatrix} \text{CAT}: & s \end{bmatrix}$$

The single element of this AMRS unifies with the head of the first rule in the grammar, trivially. Substitution is again trivial, and the next form in the derivation is the body of the first rule:

$$\sigma_2 = \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{1} \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \end{bmatrix}$$

Derivation

Example (Derivation)

Since the rule ρ of that example is indeed in G_u, a derivable form from σ_2 is:

$$\sigma_3 = \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & \boxed{4} \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{4} \end{bmatrix}$$

Thus, we obtain $\sigma_1 \Rightarrow \sigma_2 \Rightarrow \sigma_3$, and hence $\sigma_1 \overset{*}{\Rightarrow} \sigma_3$.

Derivation

Example (Derivation)

Consider the form σ_3 and one of the AMRSs in $PT_w(1,3)$:

$$\sigma_3 = \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & \boxed{4} \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{4} \end{bmatrix}$$

$$\sigma = \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & pl \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & [\,] \end{bmatrix} \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & pl \end{bmatrix}$$

The former contains information that is accumulated during derivations; the latter reflect information from the lexical entries of the words in w.

$$\sigma \sqcup \rho = \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & \boxed{4}pl \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{4} \end{bmatrix}$$

Language

Definition (Language)

The **language** of a unification grammar G is
$L(G) = \{w \in \textsc{Words}^* \mid w = w_1 \cdots w_n$ and there exist an AMRS σ such that $A_s \overset{*}{\Rightarrow} \sigma$ and an AMRS $\rho \in PT_w(1, n)$ such that $\sigma \sqcup \rho$ is defined$\}$.

Language

Example (Language)

Consider the grammar G_u and the string the sheep sleep. The form σ_3 is derivable from the start symbol of the grammar. This form is unifiable with one of the members of $PT_w(1,3)$. Hence the string the sheep sleep is a member of $L(G_u)$.

Derivation trees

- In order to depict derivations graphically we extend the notion of *derivation trees*, defined for context-free grammars, to unification grammars
- Informally, we would like a tree to be a structure whose elements are feature structures
- However, care must be taken when the scope of reentrancies in a tree is concerned: in order for information to be shared among all nodes in a tree, this scope is extended to the entire tree

Derivation trees

- Rather than define a new mathematical entity, corresponding to a tree whose nodes are feature structures with the scope of reentrancies extended to the entire structure, we reuse in the following definition the concept of multi-rooted structures (more precisely, AMRSs)
- In order to impose a tree structure on AMRSs we simply pair them with a tree whose nodes are integers, such that each node in the tree serves as an index into the AMRS
- In this way, all the existing definitions which refer to AMRSs can be naturally used when reasoning about trees

Derivation trees

Definition (Unification trees)

Given a signature $\mathcal{S} = \langle \textsc{Atoms}, \textsc{Feats} \rangle$, a **unification tree** is an ordered tree whose nodes are AVMs over $\mathcal{S}$, where the scope of reentrancies is extended to the entire tree. A **subtree** is a particular node of the tree, along with all its descendants (and the edges connecting them). Formally, a unification tree is a pair $\langle \sigma, \tau \rangle$, where σ is an AMRS over $\mathcal{S}$, say of length l for some $l \in \mathbb{N}$, and τ is a tree over the nodes $\{1, 2, \ldots, l\}$.

Derivation trees

Example (Unification tree)

Following is a unification tree, depicted as a tree of AVMs:

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

$$\begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}nom\end{bmatrix}$$

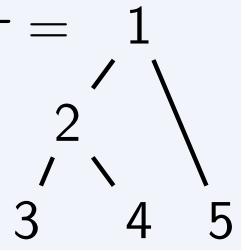
$$\begin{bmatrix}\text{CAT}: & d \\ \text{NUM}: & \boxed{4}\end{bmatrix} \quad \begin{bmatrix}\text{CAT}: & n \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}nom\end{bmatrix} \quad \begin{bmatrix}\text{CAT}: & v \\ \text{NUM}: & \boxed{4}\end{bmatrix}$$

Derivation trees

Example (Unification tree)

Formally, this tree is a pair $\langle \tau, \sigma \rangle$, where τ is a tree over $\{1, 2, 3, 4, 5\}$ and σ is an AMRS of length 5:

$\tau =$



$$\sigma = \begin{bmatrix} \text{CAT}: & s \end{bmatrix} \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & \boxed{4} \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{4} \end{bmatrix}$$

Unification derivation trees

Definition (Unification derivation trees)

A **unification derivation tree** induced by a unification grammar $G = (\mathcal{R}, A^s)$ is a unification tree defined recursively as follows:

- $\langle A^s, \tau \rangle$ is a unification derivation tree, where τ is the tree consisting of the single node $\{1\}$;
- if $\langle \sigma, \tau \rangle$ is a unification derivation tree and $\langle \sigma', \tau' \rangle$ extends $\langle \sigma, \tau \rangle$, then $\langle \sigma', \tau' \rangle$ is also a unification derivation tree.

Unification derivation trees

Example (Unification derivation trees)

A unification derivation tree with the grammar G_u can be built incrementally as follows. The start symbol of the grammar is $\begin{bmatrix}\text{CAT}: & s\end{bmatrix}$; therefore, an initial derivation tree would be $\langle \sigma_1, \{1\} \rangle$, the start symbol itself.

Then, by using the first grammar rule, the following tree, $\langle \sigma_2, \tau_2 \rangle$, can be obtained:

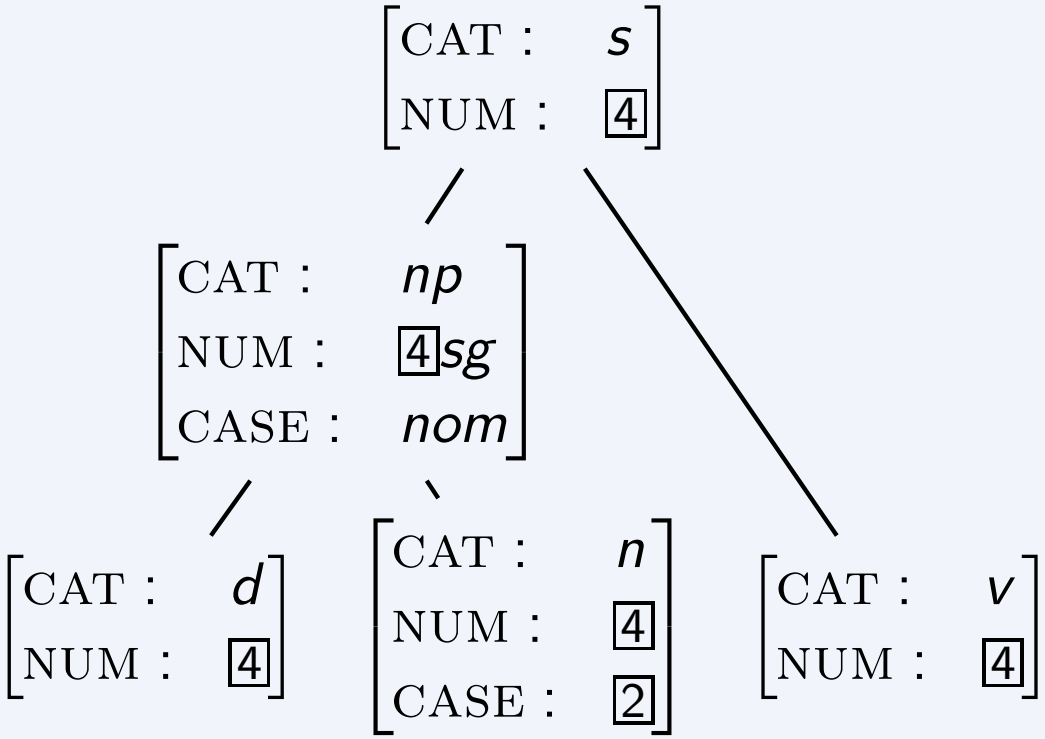
$$\begin{bmatrix}\text{CAT}: & s \\ \text{NUM}: & \boxed{4}\end{bmatrix}$$

$$\begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & nom\end{bmatrix} \qquad \begin{bmatrix}\text{CAT}: & v \\ \text{NUM}: & \boxed{4}\end{bmatrix}$$

Unification derivation trees

Example (Unification derivation trees)

Next, by applying the second grammar rule to the leftmost node on the frontier of $\langle \sigma_2, \tau_2 \rangle$, the following tree, $\langle \sigma_3, \tau_3 \rangle$, is obtained:



Complete derivation trees

- As in the context-free case, the frontier of unification derivation trees does not have to correspond to any lexical item
- Of course, in order for trees to represent complete derivations, we are particularly interested in such trees whose frontier is unifiable with a sequence of pre-terminals

Complete derivation trees

Definition (Complete derivation trees)

A unification derivation tree $\langle \sigma, \tau \rangle$ is **complete** if the frontier of τ is $j_1, \ldots, j_n$ and there exist a word $w \in \textsc{Words}^*$ of length n and an AMRS $\rho \in PT_w(1, n)$ such that $\rho \sqcup \langle \sigma_i, \sigma_{j_1}, \ldots, \sigma_{j_n} \rangle$ is defined.

Note that there may be more than one qualifying AMRS in $PT_w(1, n)$; the definition only requires one. Of course, different AMRSs in $PT_w(1, n)$ will correspond to different interpretations of the input string (resulting from ambiguous lexical entries of the words)

Complete derivation trees

Example (Complete derivation trees)

Consider the grammar G_u and the string $w =$ two lambs sleep. The tree of the previous example is complete. Its frontier is unifiable with the following AMRS:

$$\begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & pl \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & \boxed{2} \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & pl \end{bmatrix} \in PT_w(1,3)$$

Lexicalized derivation trees

- It is sometimes useful to depict a tree whose leaves already reflect the additional information obtained by actually unifying the frontier of a complete derivation tree with PT_w
- We call such trees *lexicalized*
- It is easy to see that for every lexicalized tree $\langle \sigma, \tau \rangle$ there exists a complete derivation tree $\langle \sigma', \tau' \rangle$ such that $\tau' = \tau$ and $\sigma' \overset{\hat{\rightarrow}}{\sqsubseteq} \sigma$

Lexicalized derivation trees

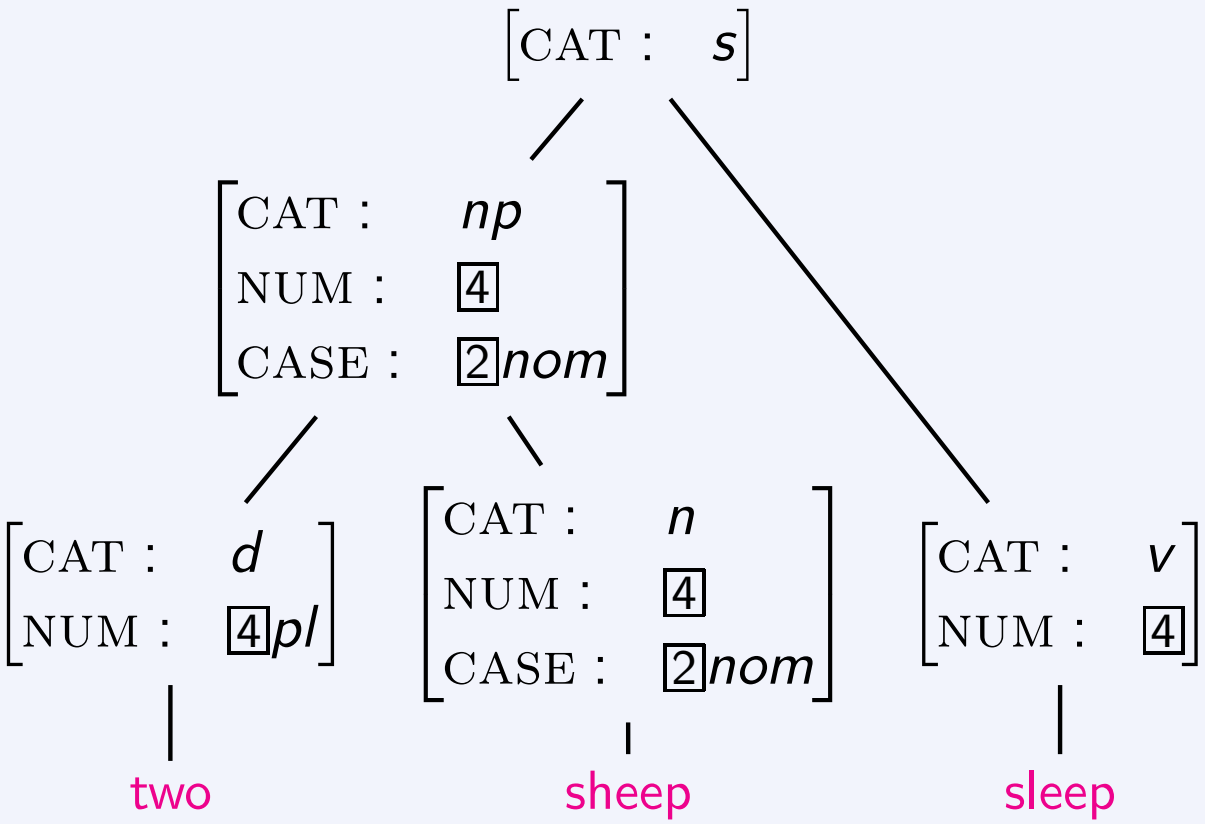
Definition (Lexicalized derivation trees)

Let $\langle \sigma, \tau \rangle$ be a complete derivation tree induced by a unification grammar $G = (\mathcal{R}, A^s)$ and let w, ρ be as in the definition of complete trees. A **lexicalized derivation tree** induced by G on w is the unification tree $\langle \sigma', \tau \rangle$, where σ' is obtained from σ by unifying the frontier of σ with ρ.

Lexicalized derivation trees

Example (Lexicalized derivation tree)

A tree induced by the grammar G_u on the string two lambs sleep:



Linguistic applications

- We now put the theory to use, by accounting for several of the linguistic phenomena that motivated UGs
- Unification grammars facilitate the expression of linguistic generalizations
- This is mediated through two main mechanisms:
  - The notion of grammatical category is expressed via feature structures, thereby allowing for complex categories as first-class citizens of the grammatical theory
  - Reentrancy provides a concise machinery for expressing "movement", or more generally, relations that hold in a deeper level than a phrase-structure tree

Phenomena

- Agreement
- Case control
- Subcategorization
- Long-distance dependencies
- Control
- Coordination

A basic grammar

Example (A context-free grammar G_0:)

S	$\rightarrow$	$NP\ VP$
VP	$\rightarrow$	$V \mid V\ NP$
NP	$\rightarrow$	$D\ N \mid Pron \mid PropN$
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, ...

Every CFG is a UG

- Observe that *any* context-free grammar is a special case of a unification grammar
- The non-terminal symbols of the CFG can be modeled by atoms
- A more general view of G_0 as a unification grammar can encode the fact that the non-terminal symbols represent grammatical categories
- This can be done using a single feature, e.g., CAT, whose values are the non-terminals of G_0

G_0', a basic unification grammar

Example (G_0', a basic unification grammar)

Following is a unification grammar, G_0', over a signature $\langle \textsc{Feats}, \textsc{Atoms} \rangle$ where $\textsc{Feats} = \{\textsc{cat}\}$ and $\textsc{Atoms} = \{s, np, vp, v, d, n, pron, propn\}$:

$$
\begin{array}{lllll}
1 & \begin{bmatrix}\textsc{cat}: & s\end{bmatrix} & \rightarrow & \begin{bmatrix}\textsc{cat}: & np\end{bmatrix} \begin{bmatrix}\textsc{cat}: & vp\end{bmatrix} \\
2 & \begin{bmatrix}\textsc{cat}: & vp\end{bmatrix} & \rightarrow & \begin{bmatrix}\textsc{cat}: & v\end{bmatrix} \\
3 & \begin{bmatrix}\textsc{cat}: & vp\end{bmatrix} & \rightarrow & \begin{bmatrix}\textsc{cat}: & v\end{bmatrix} \begin{bmatrix}\textsc{cat}: & np\end{bmatrix} \\
4 & \begin{bmatrix}\textsc{cat}: & np\end{bmatrix} & \rightarrow & \begin{bmatrix}\textsc{cat}: & d\end{bmatrix} \begin{bmatrix}\textsc{cat}: & n\end{bmatrix} \\
5, 6 & \begin{bmatrix}\textsc{cat}: & np\end{bmatrix} & \rightarrow & \begin{bmatrix}\textsc{cat}: & pron\end{bmatrix} \mid \begin{bmatrix}\textsc{cat}: & propn\end{bmatrix}
\end{array}
$$

G_0', a basic unification grammar

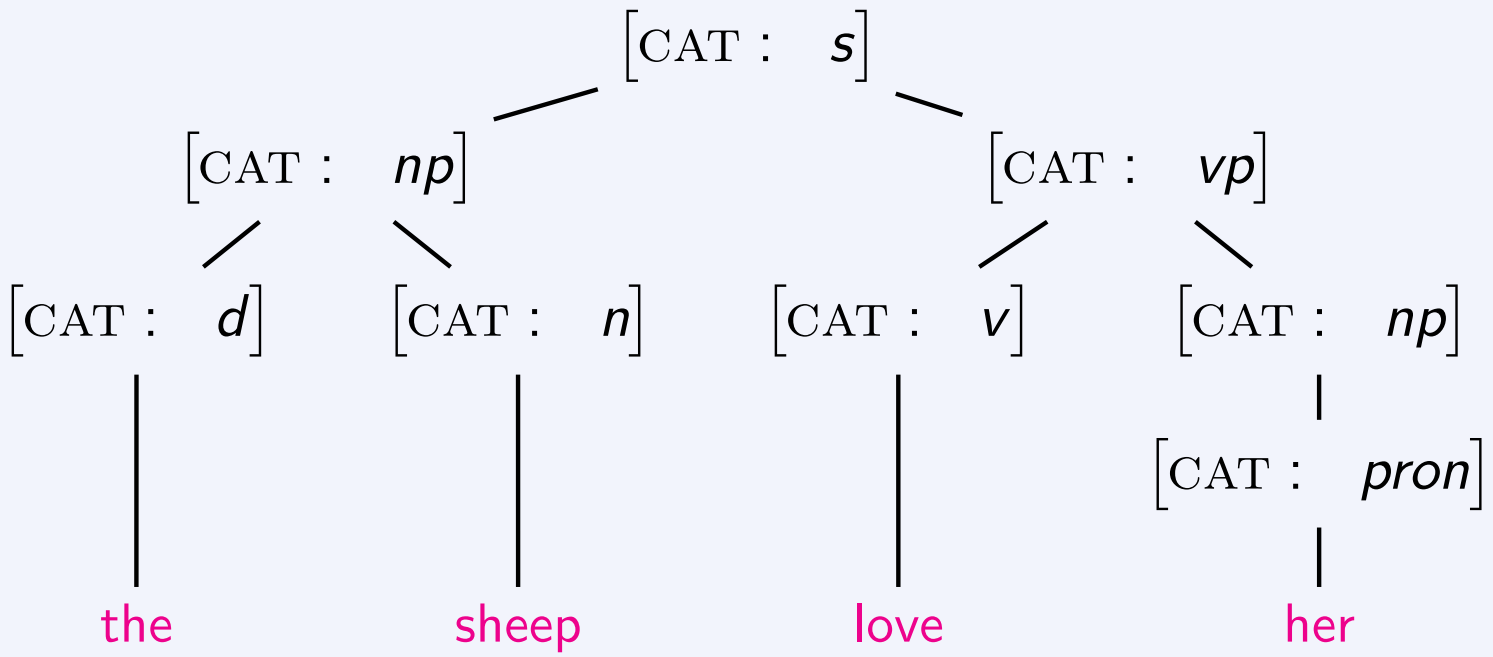
Example (G_0', a basic unification grammar)

sleep	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & v\end{bmatrix}$	give	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & v\end{bmatrix}$
love	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & v\end{bmatrix}$	tell	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & v\end{bmatrix}$
feed	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & v\end{bmatrix}$	feeds	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & v\end{bmatrix}$
lamb	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & n\end{bmatrix}$	lambs	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & n\end{bmatrix}$
she	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & pron\end{bmatrix}$	her	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & pron\end{bmatrix}$
they	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & pron\end{bmatrix}$	them	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & pron\end{bmatrix}$
Rachel	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & propn\end{bmatrix}$	Jacob	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & propn\end{bmatrix}$
a	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & d\end{bmatrix}$	two	$\longrightarrow$	$\begin{bmatrix}\text{CAT}: & d\end{bmatrix}$

Derivation trees induced by G_0'

Example (Derivation trees induced by G_0')

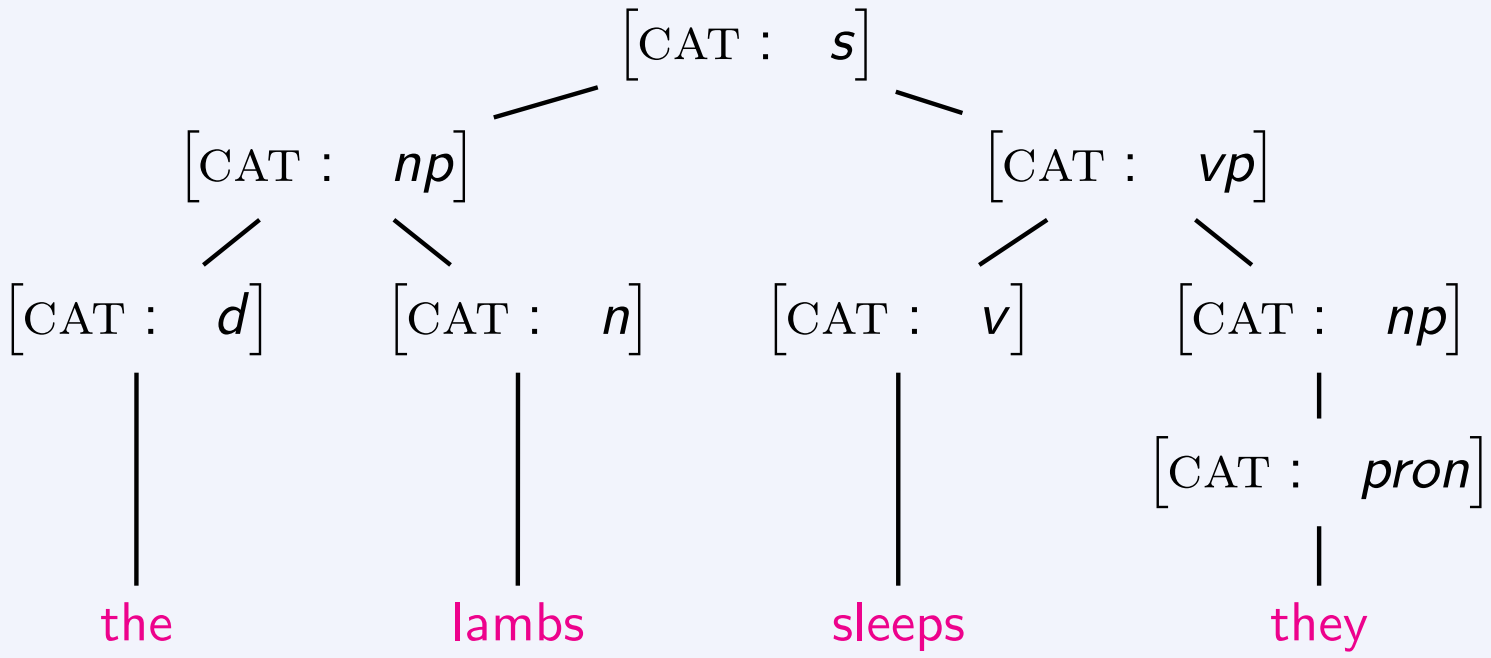
The grammar G_0' induces the following tree on the string the sheep love her:



Derivation trees induced by G_0'

Example (Derivation trees induced by G_0')

Not surprisingly, an isomorphic derivation tree is induced by the grammar on the ungrammatical string ∗the lambs sleeps they:



G_{agr}, accounting for agreement on number

Example (G_{agr}, accounting for agreement on number)

1	$\begin{bmatrix}\text{CAT}: & s\end{bmatrix}$	$\rightarrow$	$\begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4}\end{bmatrix}\ \begin{bmatrix}\text{CAT}: & vp \\ \text{NUM}: & \boxed{4}\end{bmatrix}$
2	$\begin{bmatrix}\text{CAT}: & vp \\ \text{NUM}: & \boxed{4}\end{bmatrix}$	$\rightarrow$	$\begin{bmatrix}\text{CAT}: & v \\ \text{NUM}: & \boxed{4}\end{bmatrix}$
3	$\begin{bmatrix}\text{CAT}: & vp \\ \text{NUM}: & \boxed{4}\end{bmatrix}$	$\rightarrow$	$\begin{bmatrix}\text{CAT}: & v \\ \text{NUM}: & \boxed{4}\end{bmatrix}\ \begin{bmatrix}\text{CAT}: & np\end{bmatrix}$
4	$\begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4}\end{bmatrix}$	$\rightarrow$	$\begin{bmatrix}\text{CAT}: & d \\ \text{NUM}: & \boxed{4}\end{bmatrix}\ \begin{bmatrix}\text{CAT}: & n \\ \text{NUM}: & \boxed{4}\end{bmatrix}$
5, 6	$\begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4}\end{bmatrix}$	$\rightarrow$	$\begin{bmatrix}\text{CAT}: & pron \\ \text{NUM}: & \boxed{4}\end{bmatrix} \mid \begin{bmatrix}\text{CAT}: & propn \\ \text{NUM}: & \boxed{4}\end{bmatrix}$

G_{agr}, accounting for agreement on number

Example (G_{agr}, accounting for agreement on number)

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

$$\text{love} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & pl \end{bmatrix} \qquad \text{tell} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & pl \end{bmatrix}$$

$$\text{feed} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & pl \end{bmatrix} \qquad \text{feeds} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & sg \end{bmatrix}$$

G_{agr}, accounting for agreement on number

Example (G_{agr}, accounting for agreement on number)

lamb	$\rightarrow$	$\begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & sg \end{bmatrix}$	lambs	$\rightarrow$	$\begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \end{bmatrix}$
she	$\rightarrow$	$\begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & sg \end{bmatrix}$	her	$\rightarrow$	$\begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & sg \end{bmatrix}$
they	$\rightarrow$	$\begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & pl \end{bmatrix}$	them	$\rightarrow$	$\begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & pl \end{bmatrix}$
Rachel	$\rightarrow$	$\begin{bmatrix} \text{CAT}: & propn \\ \text{NUM}: & sg \end{bmatrix}$	Jacob	$\rightarrow$	$\begin{bmatrix} \text{CAT}: & propn \\ \text{NUM}: & sg \end{bmatrix}$
a	$\rightarrow$	$\begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & sg \end{bmatrix}$	two	$\rightarrow$	$\begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & pl \end{bmatrix}$

G_{agr} generates a CF language

Example (A context-free grammar G_1)

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

$S_{sg} \rightarrow NP_{sg}\ VP_{sg}$

$NP_{sg} \rightarrow D_{sg}\ N_{sg}$

$NP_{sg} \rightarrow Pron_{sg} \mid PropN_{sg}$

$VP_{sg} \rightarrow V_{sg}$

$VP_{sg} \rightarrow V_{sg}\ NP_{sg} \mid V_{sg}\ NP_{pl}$

$D_{sg} \rightarrow a$

$N_{sg} \rightarrow lamb \mid sheep \mid \cdots$

$Pron_{sg} \rightarrow she \mid her \mid \cdots$

$V_{sg} \rightarrow sleeps \mid \cdots$

$S_{pl} \rightarrow NP_{pl}\ VP_{pl}$

$NP_{pl} \rightarrow D_{pl}\ N_{pl}$

$NP_{pl} \rightarrow Pron_{pl} \mid PropN_{pl}$

$VP_{pl} \rightarrow V_{pl}$

$VP_{pl} \rightarrow V_{pl}\ NP_{sg} \mid V_{pl}\ NP_{pl}$

$D_{pl} \rightarrow two$

$N_{pl} \rightarrow lambs \mid sheep \mid \cdots$

$PropN_{sg} \rightarrow Rachel \mid Jacob \mid \cdots$

$V_{pl} \rightarrow sleep \mid \cdots$

G_{agr} generates a CF language

- While G_{agr} is a unification grammar, the language it generates *is* context free
- But the equivalent CFG is inferior to the unification grammar:
  - The linguistic description is distorted: information regarding NUMBER, which is determined by the words themselves, is encoded in G_1 by the way they are derived (in other words, G_1 accounts for lexical knowledge by means of phrase-structure rules)
  - Several linguistic generalizations are lost: the context-free grammar induces two different trees on a lamb sleeps and two lambs sleep

UG and linguistic generalizations

- One natural notion of 'linguistic generalization' emerges: the ability to formulate a linguistic restriction by means of a single rule, instead of by a collection of "similar" rules
- In this sense, G_{agr} captures the agreement generalization, while G_1 does not
- Multiplying out all the possible values of a particular feature, and converting a unification grammar to an equivalent context-free grammar in this way, is not always possible

Imposing case control

- Add a feature to the signature, CASE, to the feature structures associated with nominal categories: nouns, pronouns, proper names and noun phrases
- The lexical entries of pronouns must specify their case, which is overt and explicit: we use the value *nom* for nominative case, whereas *acc* stands for accusative
- As for proper names and nouns, their lexical entries are simply underspecified with respect to case
- Use the values of the CASE feature in the grammar to impose constraints of case control

G_{case}, accounting for case control

Example (G_{case}, accounting for case control)

1 $\begin{bmatrix} \text{CAT}: & s \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & vp \\ \text{NUM}: & \boxed{4} \end{bmatrix}$

2 $\begin{bmatrix} \text{CAT}: & vp \\ \text{NUM}: & \boxed{4} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{4} \end{bmatrix}$

3 $\begin{bmatrix} \text{CAT}: & vp \\ \text{NUM}: & \boxed{4} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{4} \end{bmatrix} \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{3} \\ \text{CASE}: & acc \end{bmatrix}$

4 $\begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & d \\ \text{NUM}: & \boxed{4} \end{bmatrix} \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2} \end{bmatrix}$

5, 6 $\begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2} \end{bmatrix} \mid \begin{bmatrix} \text{CAT}: & propn \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2} \end{bmatrix}$

G_{case}, accounting for case control

Example (G_{case}, accounting for case control)

$$\text{sleep} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & pl \end{bmatrix} \qquad \text{sleeps} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & sg \end{bmatrix}$$

$$\text{feed} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & pl \end{bmatrix} \qquad \text{feeds} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & sg \end{bmatrix}$$

G_{case}, accounting for case control

Example (G_{case}, accounting for case control)

$$\text{lamb} \rightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & sg \\ \text{CASE}: & [\,] \end{bmatrix} \qquad \text{lambs} \rightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & [\,] \end{bmatrix}$$

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

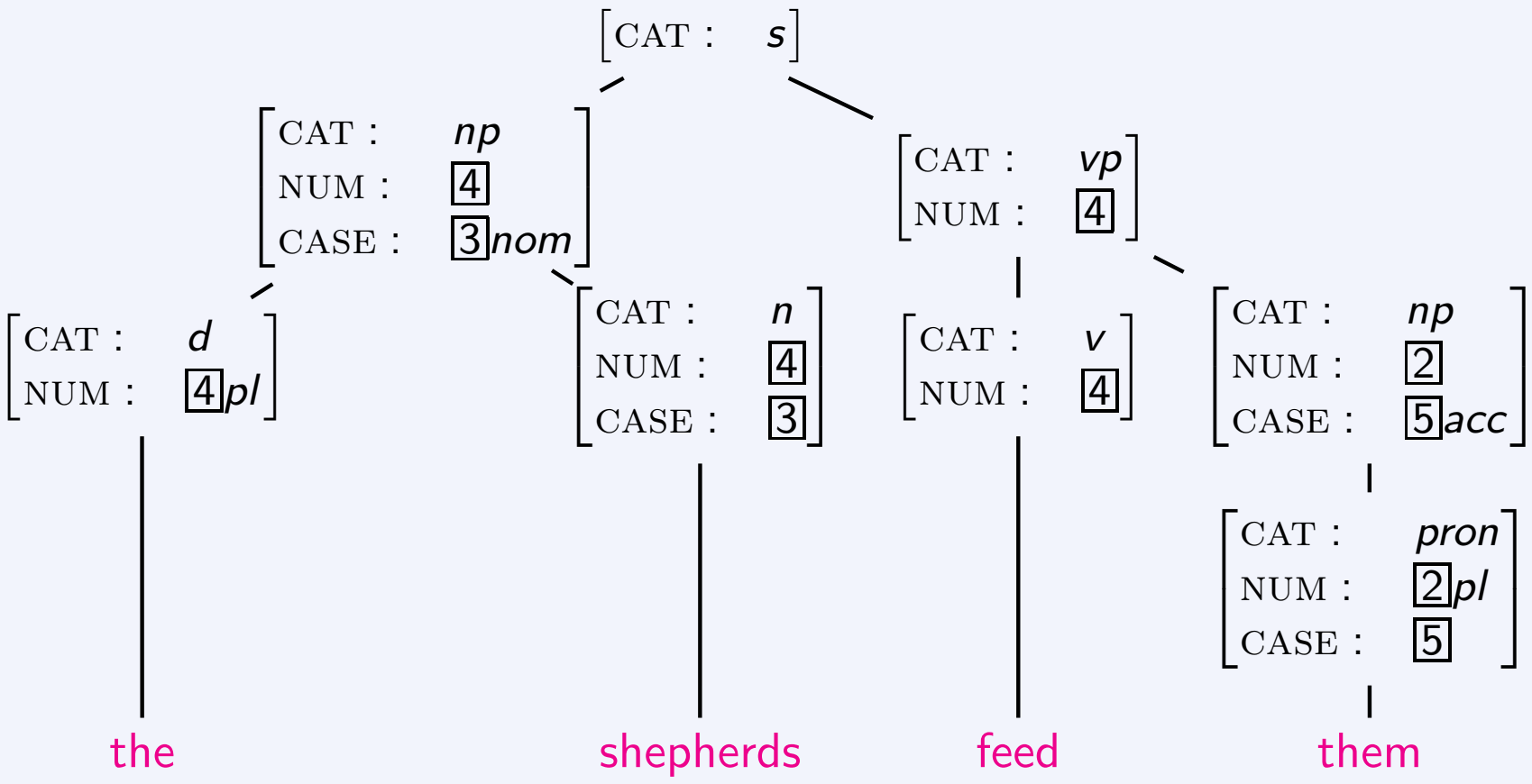
$$\text{they} \rightarrow \begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & pl \\ \text{CASE}: & nom \end{bmatrix} \qquad \text{them} \rightarrow \begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & pl \\ \text{CASE}: & acc \end{bmatrix}$$

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

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

Derivation tree with case control

Example (Derivation tree with case control)



Derivation tree with case control

Example (Derivation tree with case control)

This tree represents a derivation which starts with the initial symbol, $\begin{bmatrix}\text{CAT}: & s\end{bmatrix}$, and ends with multi-AVM σ', where

$$\sigma' = \overset{\textit{the}}{\begin{bmatrix}\text{NUM}: & \boxed{4}\end{bmatrix}} \overset{\textit{shepherds}}{\begin{bmatrix}\text{NUM}: & \boxed{4} \\ \text{CASE}: & \textit{nom}\end{bmatrix}} \overset{\textit{feed}}{\begin{bmatrix}\text{NUM}: & \boxed{4}\end{bmatrix}} \overset{\textit{them}}{\begin{bmatrix}\text{NUM}: & \boxed{2} \\ \text{CASE}: & \textit{acc}\end{bmatrix}}$$

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

$$\sigma = \overset{\textit{the}}{\begin{bmatrix}\text{NUM}: & [\,]\end{bmatrix}} \overset{\textit{shepherds}}{\begin{bmatrix}\text{NUM}: & \textit{pl} \\ \text{CASE}: & [\,]\end{bmatrix}} \overset{\textit{feed}}{\begin{bmatrix}\text{NUM}: & \textit{pl}\end{bmatrix}} \overset{\textit{them}}{\begin{bmatrix}\text{NUM}: & \textit{pl} \\ \text{CASE}: & \textit{acc}\end{bmatrix}}$$

Hence the sentence is in the language generated by the grammar.

Imposing subcategorization constraints

- A naïve solution to the subcategorization problem

  intransitive verbs (with no object): sleep, walk, run, laugh, ...
  transitive verbs (with a nominal object): feed, love, eat, ...
- Lexical entries of verbs are extended such that their subcategorization is specified
- The rules that involve verbs and verb phrases are extended

G_{subcat}, a naïve account of verb subcategorization

Example (G_{subcat}, a naïve account of verb subcategorization)

1 $$\begin{bmatrix}\text{CAT}: & s\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & nom\end{bmatrix} \begin{bmatrix}\text{CAT}: & vp \\ \text{NUM}: & \boxed{4}\end{bmatrix}$$

2 $$\begin{bmatrix}\text{CAT}: & vp \\ \text{NUM}: & \boxed{4}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & v \\ \text{NUM}: & \boxed{4} \\ \text{SUBCAT}: & intrans\end{bmatrix}$$

3 $$\begin{bmatrix}\text{CAT}: & vp \\ \text{NUM}: & \boxed{4}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & v \\ \text{NUM}: & \boxed{4} \\ \text{SUBCAT}: & trans\end{bmatrix} \begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & acc\end{bmatrix}$$

4 $$\begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & d \\ \text{NUM}: & \boxed{4}\end{bmatrix} \begin{bmatrix}\text{CAT}: & n \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix}$$

5, 6 $$\begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & pron \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix} \mid \begin{bmatrix}\text{CAT}: & propn \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix}$$

G_{subcat}, a naïve account of verb subcategorization

Example (G_{subcat}, a naïve account of verb subcategorization)

$$\text{sleep} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & pl \\ \text{SUBCAT}: & intrans \end{bmatrix} \qquad \text{sleeps} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & sg \\ \text{SUBCAT}: & intrans \end{bmatrix}$$

$$\text{feed} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & pl \\ \text{SUBCAT}: & trans \end{bmatrix} \qquad \text{feeds} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & sg \\ \text{SUBCAT}: & trans \end{bmatrix}$$

G_{subcat}, a naïve account of verb subcategorization

Example (G_{subcat}, a naïve account of verb subcategorization)

lamb $\rightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & sg \\ \text{CASE}: & [\,] \end{bmatrix}$ lambs $\rightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & [\,] \end{bmatrix}$

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

they $\rightarrow \begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & pl \\ \text{CASE}: & nom \end{bmatrix}$ them $\rightarrow \begin{bmatrix} \text{CAT}: & pron \\ \text{NUM}: & pl \\ \text{CASE}: & acc \end{bmatrix}$

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

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

Subcategorization lists

- The previous account of subcategorization is naïve
- Different verbs *subcategorize for* different kinds of complements: noun phrases, infinitival verb phrases, sentences etc.
- Some verbs require more than one complement
- The idea is to store in the lexical entry of each verb not an atomic feature indicating its subcategory, but rather a *list* of categories, indicating the appropriate complements of the verb

Lexical entries of some verbs using subcategorization lists

Example (Lexical entries of some verbs using subcategorization lists)

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

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

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

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

Subcategorization lists

- The grammar rules must be modified to reflect the additional wealth of information in the lexical entries
- Due to this wealth there can be a dramatic reduction in the number of grammar rules necessary for handling verbs

VP rules using subcategorization lists

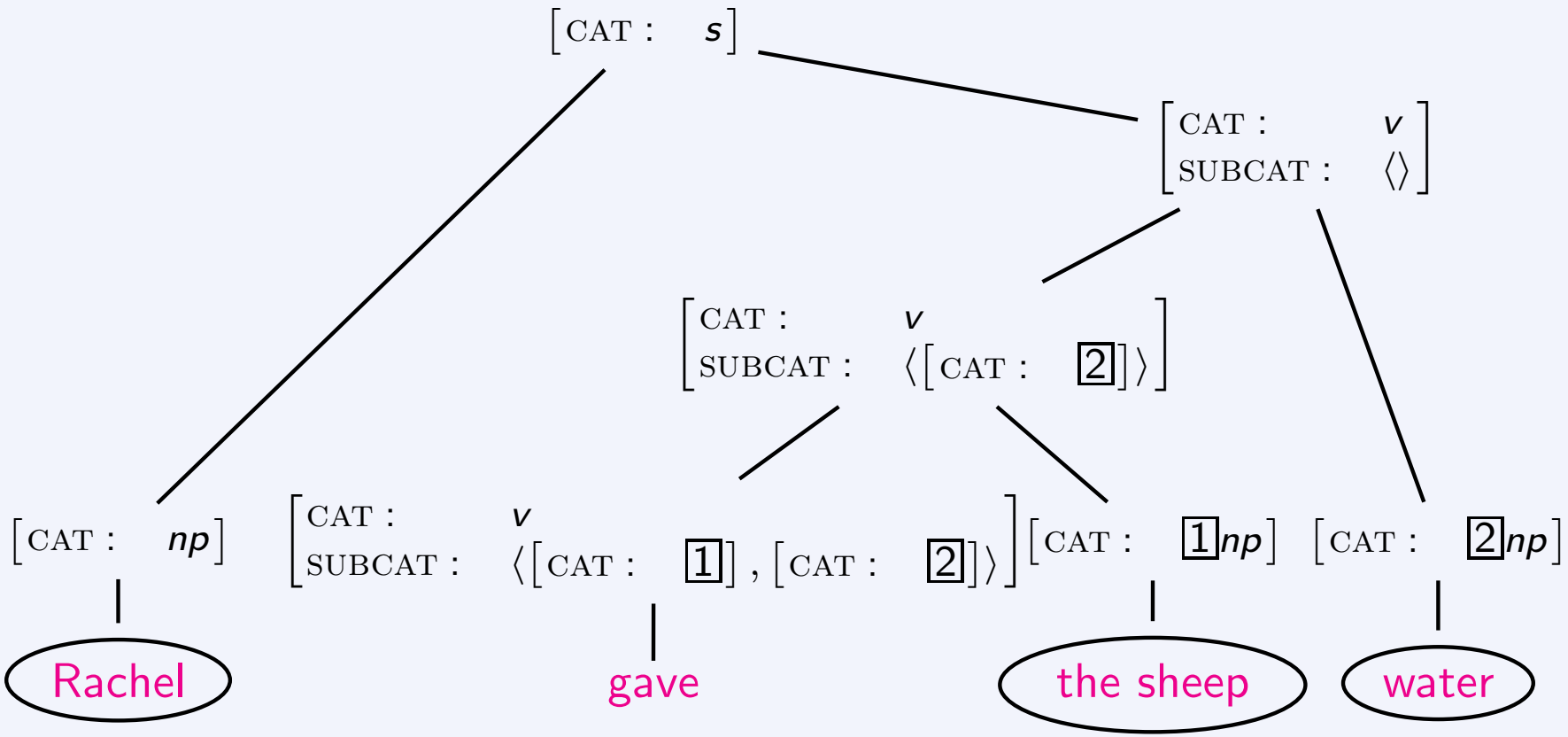
Example (VP rules using subcategorization lists)

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

$$\begin{bmatrix}\text{CAT}: & v \\ \text{SUBCAT}: & \boxed{2}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & v \\ \text{SUBCAT}: & \begin{bmatrix}\text{FIRST}: & \begin{bmatrix}\text{CAT}: & \boxed{4}\end{bmatrix} \\ \text{REST}: & \boxed{2}\end{bmatrix}\end{bmatrix} \begin{bmatrix}\text{CAT}: & \boxed{4}\end{bmatrix}$$

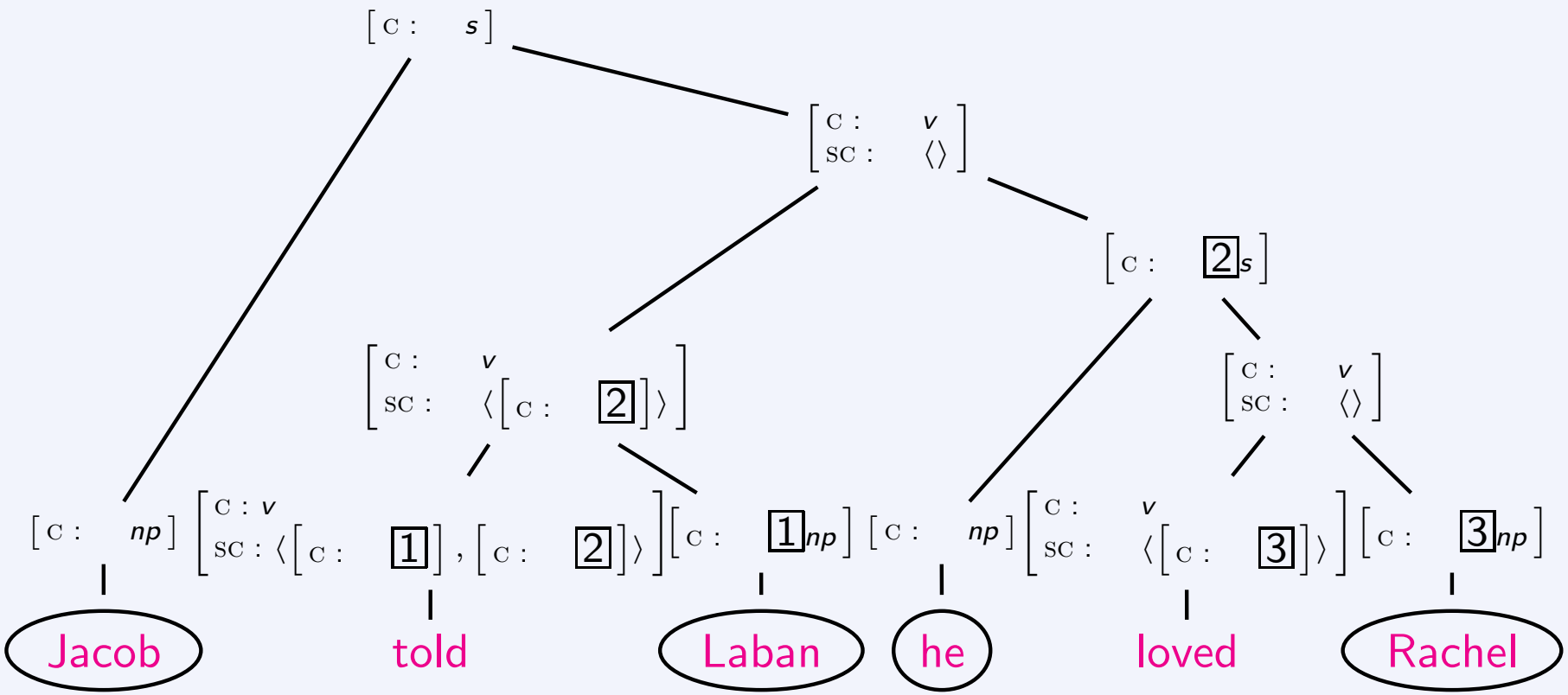
A derivation tree

Example (A derivation tree)



A derivation tree

Example (A derivation tree)



Subcategorization imposes case constraints

- In the above grammar, categories on subcategorization lists are represented as an atomic symbol
- This is a simplification; the method outlined here can be used with more complex encodings of categories
- For example, the lexical entry of the German verb geben (to give) can state that the first complement must be in the *dative* case, whereas the second must be *accusative*

Subcategorization imposes case constraints

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 imposes case constraints

Example (Subcategorization imposes case constraints)

The lexical entry of gebe, then, could be:

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

Subcategorization imposes case constraints

- In order to account for subcategorization of complex information (rather than of atomic category symbols), the VP rule which manipulates subcategorization lists has to be slightly modified
- The revised rule reflects the fact that the subcategorized information is not the *value* of the CAT feature, but rather the entire verb complement:

$$\begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{SUBCAT}: & \begin{bmatrix} \text{FIRST}: & \boxed{3} \\ \text{REST}: & \boxed{2} \end{bmatrix} \end{bmatrix} \boxed{3}[\,]$$

G_3, a complete E_2-grammar

Example (G_3, a complete E_2-grammar)

$$\begin{bmatrix}\text{CAT}: & s\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & nom\end{bmatrix} \begin{bmatrix}\text{CAT}: & v \\ \text{NUM}: & \boxed{4} \\ \text{SUBCAT}: & elist\end{bmatrix}$$

$$\begin{bmatrix}\text{CAT}: & v \\ \text{NUM}: & \boxed{4} \\ \text{SUBCAT}: & \boxed{2}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & v \\ \text{NUM}: & \boxed{4} \\ \text{SUBCAT}: & \begin{bmatrix}\text{FIRST}: & \boxed{3} \\ \text{REST}: & \boxed{2}\end{bmatrix}\end{bmatrix} \boxed{3}[\,]$$

$$\begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & d \\ \text{NUM}: & \boxed{4}\end{bmatrix} \begin{bmatrix}\text{CAT}: & n \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix}$$

$$\begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & pron \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix} \mid \begin{bmatrix}\text{CAT}: & propn \\ \text{NUM}: & \boxed{4} \\ \text{CASE}: & \boxed{2}\end{bmatrix}$$

G_3, a complete E_2-grammar

Example (G_3, a complete E_2-grammar)

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

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

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

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

G_3, a complete E_2-grammar

Example (G_3, a complete E_2-grammar)

lamb $\rightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & sg \\ \text{CASE}: & \boxed{2} \end{bmatrix}$ lambs $\rightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & \boxed{2} \end{bmatrix}$

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

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

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

Long distance dependencies

- Encoding grammatical categories as feature structures is very useful in the treatment of unbounded dependencies
- Such phenomena involve a "missing" constituent that is realized outside the clause from which it is missing, as in:

  The shepherd wondered whom Jacob loved ◡.

Long distance dependencies

- Phrases such as whom Jacob loved ◡ or who ◡ loved Rachel are *sentences*, with a constituent which is "moved" from its default position and realized as a wh-pronoun in front of the phrase
- We represent such phrases by using the category *s*
- But to distinguish them from declarative sentences we add a feature, QUE, to the category
- The value of QUE is '+' in sentences with an interrogative pronoun realizing a transposed constituent

Long distance dependencies

- We also add a lexical entry for the pronoun whom:

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

- Finally, we update the rule that derives pronouns such that it propagate the value of QUE from the lexicon to higher projections of the pronoun:

$$\begin{bmatrix} \text{CAT}: & \textit{np} \\ \text{NUM}: & \boxed{1} \\ \text{CASE}: & \boxed{3} \\ \text{QUE}: & \boxed{5} \end{bmatrix} \quad \rightarrow \quad \begin{bmatrix} \text{CAT}: & \textit{pron} \\ \text{NUM}: & \boxed{1} \\ \text{CASE}: & \boxed{3} \\ \text{QUE}: & \boxed{5} \end{bmatrix}$$

Long-distance dependencies

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

(3) $$\begin{bmatrix} \text{CAT}: & s \\ \text{SLASH}: & \boxed{4} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{1} \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \\ \text{SUBCAT}: & elist \\ \text{SLASH}: & \boxed{4} \end{bmatrix}$$

(4) $$\begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \\ \text{SUBCAT}: & \boxed{2} \\ \text{SLASH}: & \boxed{4} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \\ \text{SUBCAT}: & \begin{bmatrix} \text{FIRST}: & \boxed{4} \\ \text{REST}: & \boxed{2} \end{bmatrix} \end{bmatrix}$$

A derivation tree for Jacob loved ◡

Example (A derivation tree for Jacob loved ◡)

$$\begin{bmatrix} \text{CAT}: & s \\ \text{SLASH}: & \boxed{4} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{1} \\ \text{CASE}: & \boxed{2} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \\ \text{SLASH}: & \boxed{4} \\ \text{SUBCAT}: & \boxed{8} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & propn \\ \text{NUM}: & \boxed{1}sg \\ \text{CASE}: & \boxed{2}nom \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \\ \text{SUBCAT}: & \begin{bmatrix} \text{FIRST}: & \boxed{4}\begin{bmatrix} \text{CAT}: & np \\ \text{CASE}: & acc \end{bmatrix} \\ \text{REST}: & \boxed{8}elist \end{bmatrix} \end{bmatrix}$$

Jacob

loved

◡

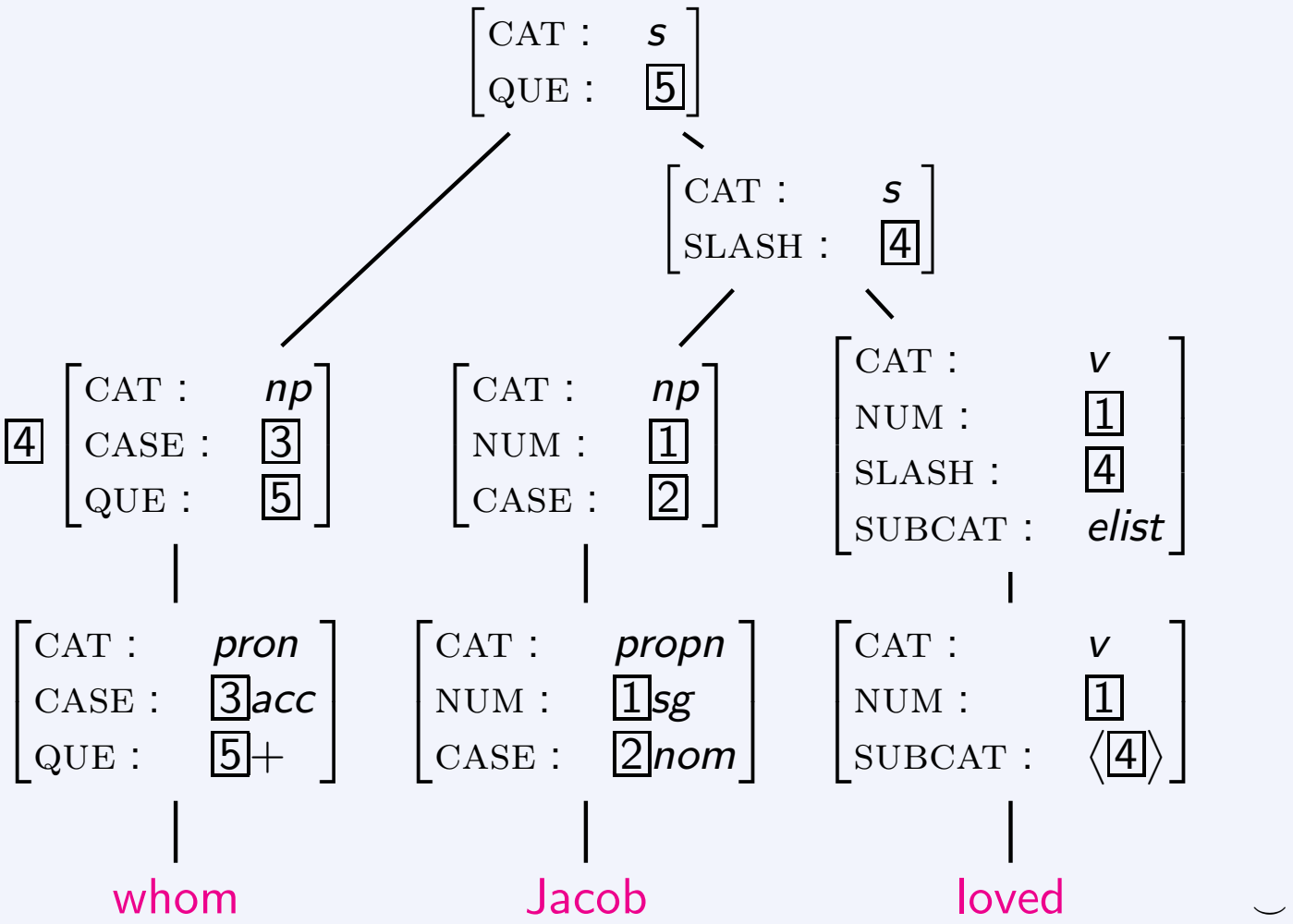
Long-distance dependencies

- A rule for creating "complete" sentences by combining the missing category with a "slashed" sentence
- The rule does not commit as to the category of the dislocated element; it simply combines *any* category with a sentence in which this very same category is missing, provided that this category is marked as 'QUE +'
- The value of QUE is propagated to the mother to indicate that the sentence is interrogative rather than declarative:

$$(5)\quad \begin{bmatrix} \text{CAT}: & s \\ \text{QUE}: & \boxed{5} \end{bmatrix} \rightarrow \boxed{4}\begin{bmatrix} \text{QUE}: & \boxed{5}+ \end{bmatrix} \begin{bmatrix} \text{CAT}: & s \\ \text{SLASH}: & \boxed{4} \end{bmatrix}$$

A derivation tree for whom Jacob loved ◡

Example (A derivation tree for whom Jacob loved ◡)



Long-distance dependencies

- In order to derive the full sentence

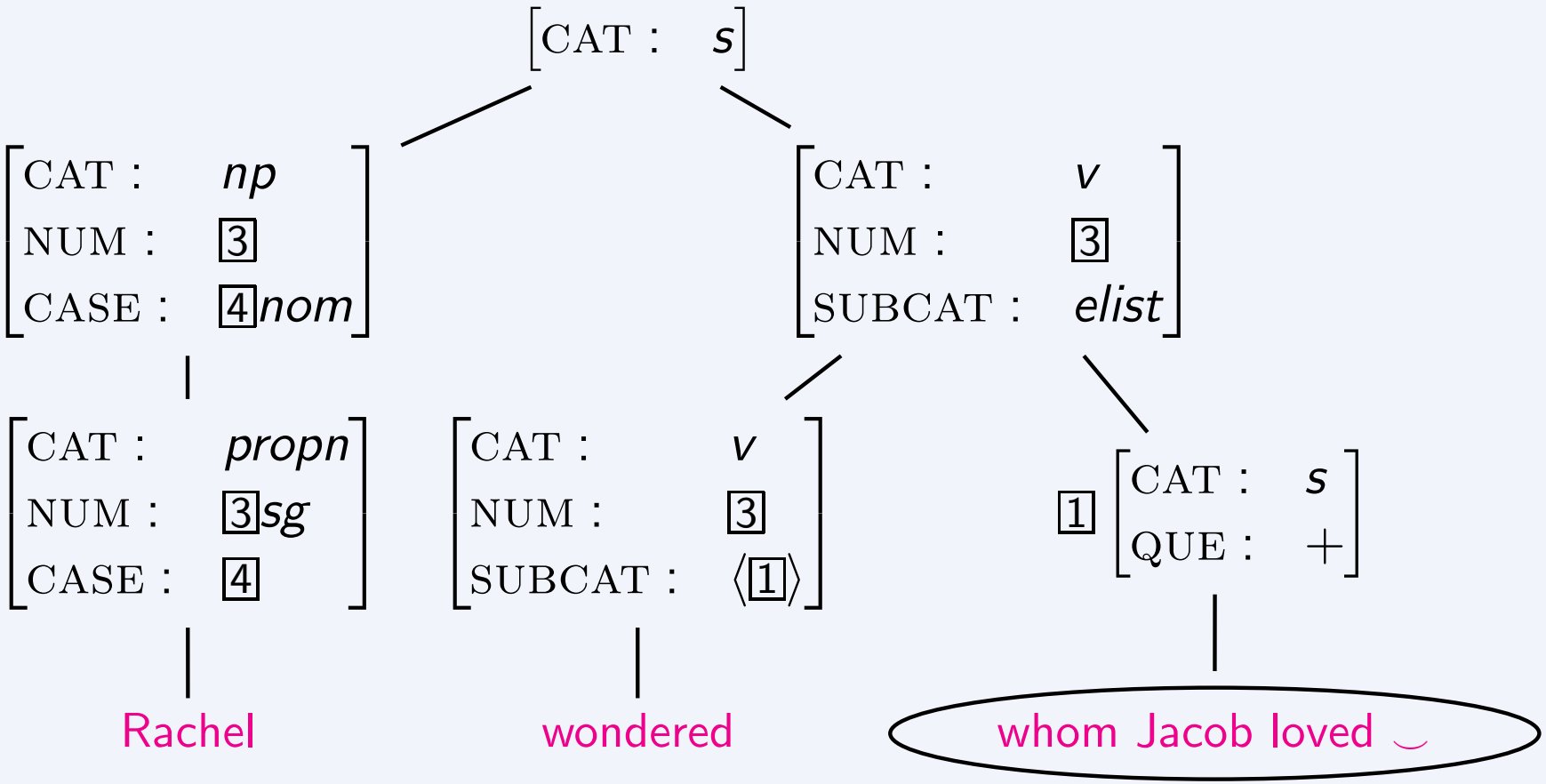
  Rachel wondered whom Jacob loved ◡

  we need a lexical entry for the verb wondered
- It is a verb, so its CATegory is *v*, and as it subcategorizes for an interrogative sentence, its SUBCATegory is a list of a single member, a sentence whose QUE feature is '+':

$$\text{wondered} \quad \rightarrow \quad \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & [\,] \\ \text{SUBCAT}: & \left\langle \begin{bmatrix} \text{CAT}: & s \\ \text{QUE}: & + \end{bmatrix} \right\rangle \end{bmatrix}$$

A derivation tree for Rachel wondered whom Jacob loved ◡

Example (A derivation tree for Rachel wondered whom Jacob loved ◡)



Long-distance dependencies

- In the previous example the filler of the gap is realized immediately to the left of the clause in which the gap occurs
- This need not always be the case: unbounded dependencies can hold across several clause boundaries
- Typical examples are:

  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

- The solution we proposed for the simple case of unbounded dependencies can be easily extended to the more complex examples
- The solution amounts to three components:
  - A slash introduction rule
  - Slash propagation rules
  - A gap filler rule

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
- 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)
- This sentence is the object of the verb thought; therefore, we need a rule that propagates the value of SLASH from a sentential object to the verb phrase of which it is an object

Long-distance dependencies

Example (Long-distance dependencies)

(6)
$$\begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \\ \text{SUBCAT}: & \boxed{12} \\ \text{SLASH}: & \boxed{4} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \\ \text{SUBCAT}: & \begin{bmatrix} \text{FIRST}: & \boxed{8} \\ \text{REST}: & \boxed{12} \end{bmatrix} \end{bmatrix} \boxed{8} \begin{bmatrix} \text{SLASH}: & \boxed{4} \end{bmatrix}$$

Long-distance dependencies

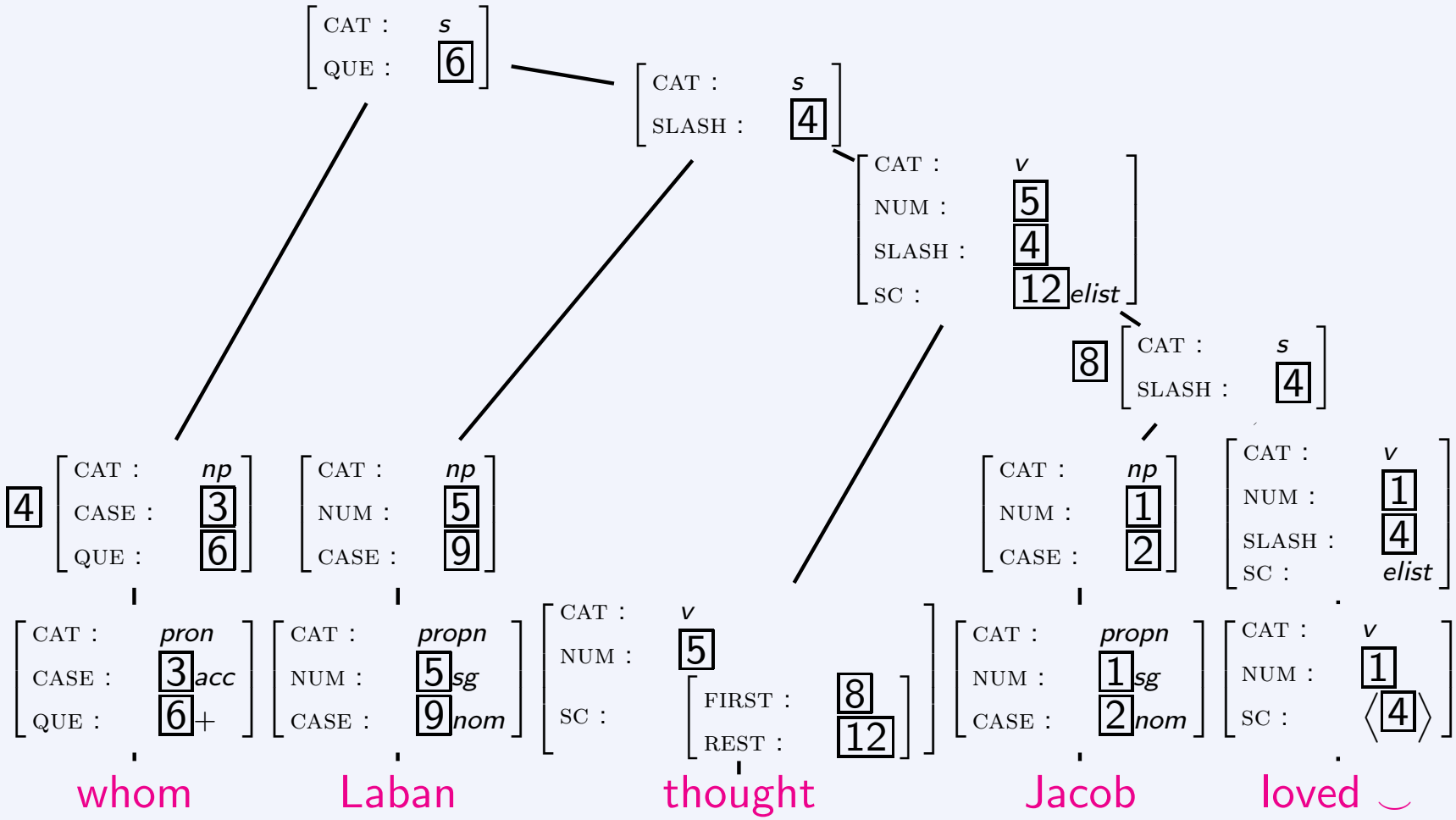
Example (Long-distance dependencies)

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

$$(7)\quad \begin{bmatrix} \text{CAT}: & s \\ \text{SLASH}: & \boxed{4} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{5} \\ \text{CASE}: & nom \end{bmatrix} \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{5} \\ \text{SUBCAT}: & elist \\ \text{SLASH}: & \boxed{4} \end{bmatrix}$$

Long-distance dependencies

Example (A derivation tree for whom Laban thought Jacob loved ◡)



Long-distance dependencies

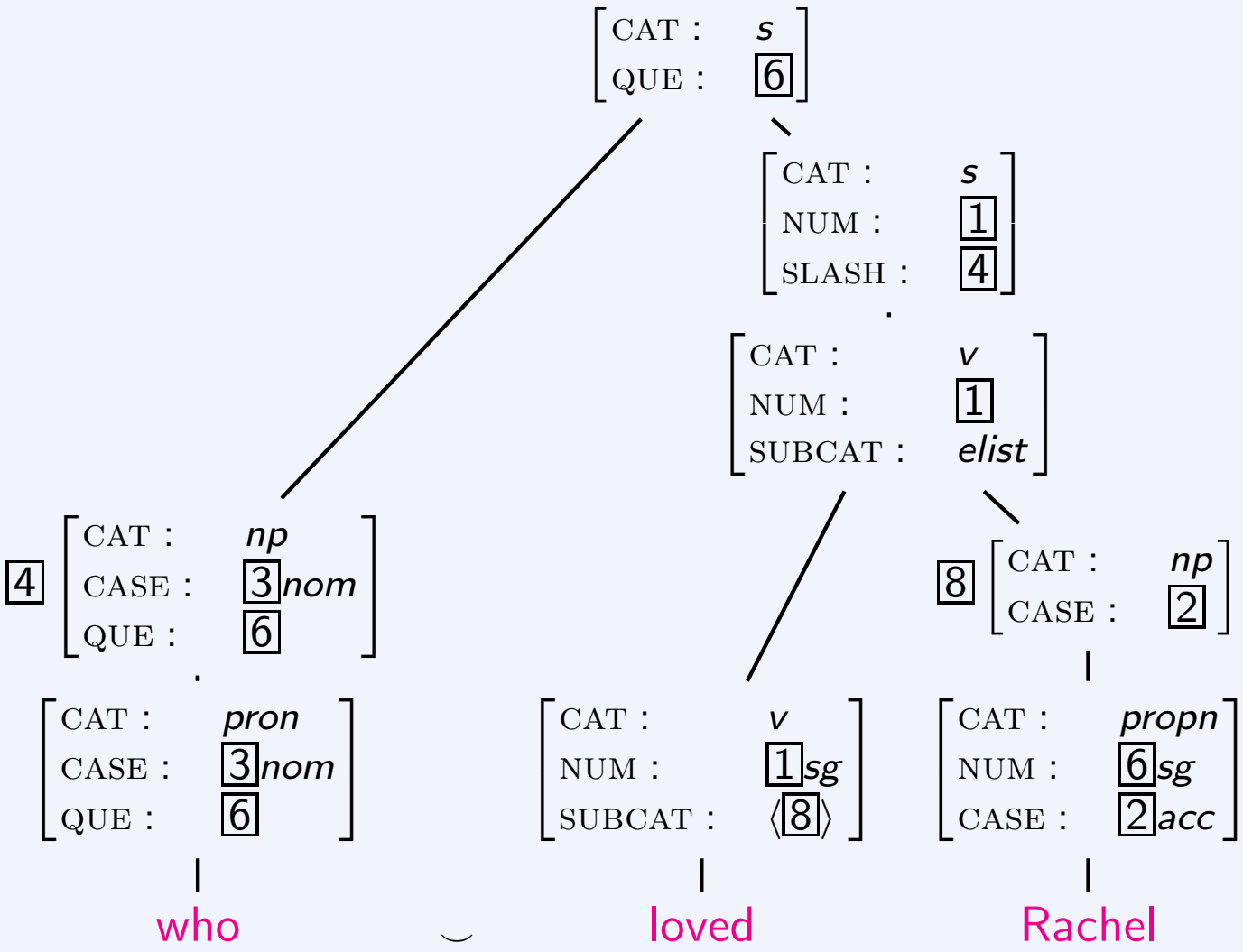
Example (Long-distance dependencies)

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

$$(8)\quad \begin{bmatrix} \text{CAT}: & s \\ \text{SLASH}: & \begin{bmatrix} \text{CAT}: & np \\ \text{NUM}: & \boxed{1} \\ \text{CASE}: & nom \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & v \\ \text{NUM}: & \boxed{1} \\ \text{SUBCAT}: & elist \end{bmatrix}$$

Long-distance dependencies

Example (A derivation tree for who ◡ loved Rachel)



Subject and object control

- 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

- The differences between the two example sentences stem from differences in the matrix verbs:
  - promise is a *subject control* verb;
  - persuade is *object control*

G_4: explicit SUBJ values

Example (G_4: explicit SUBJ values)

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

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

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

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

G_4: explicit SUBJ values

Example (G_4: explicit SUBJ values)

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

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

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

G_4: explicit SUBJ values

Example (G_4: explicit SUBJ values)

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

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

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

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

Infinitival verb phrases

- The next step is to account for infinitival verb phrases
- This can be easily done by adding a new feature, VFORM, to verbal projections
- The values of this feature can represent the form of the verb: *fin* for finite verbs and *inf* for infinitival ones
- $$\text{to work} \quad \rightarrow \quad \begin{bmatrix} \text{CAT}: & v \\ \text{VFORM}: & inf \\ \text{SUBCAT}: & elist \\ \text{SUBJ}: & \begin{bmatrix} \text{CAT}: & np \end{bmatrix} \end{bmatrix}$$

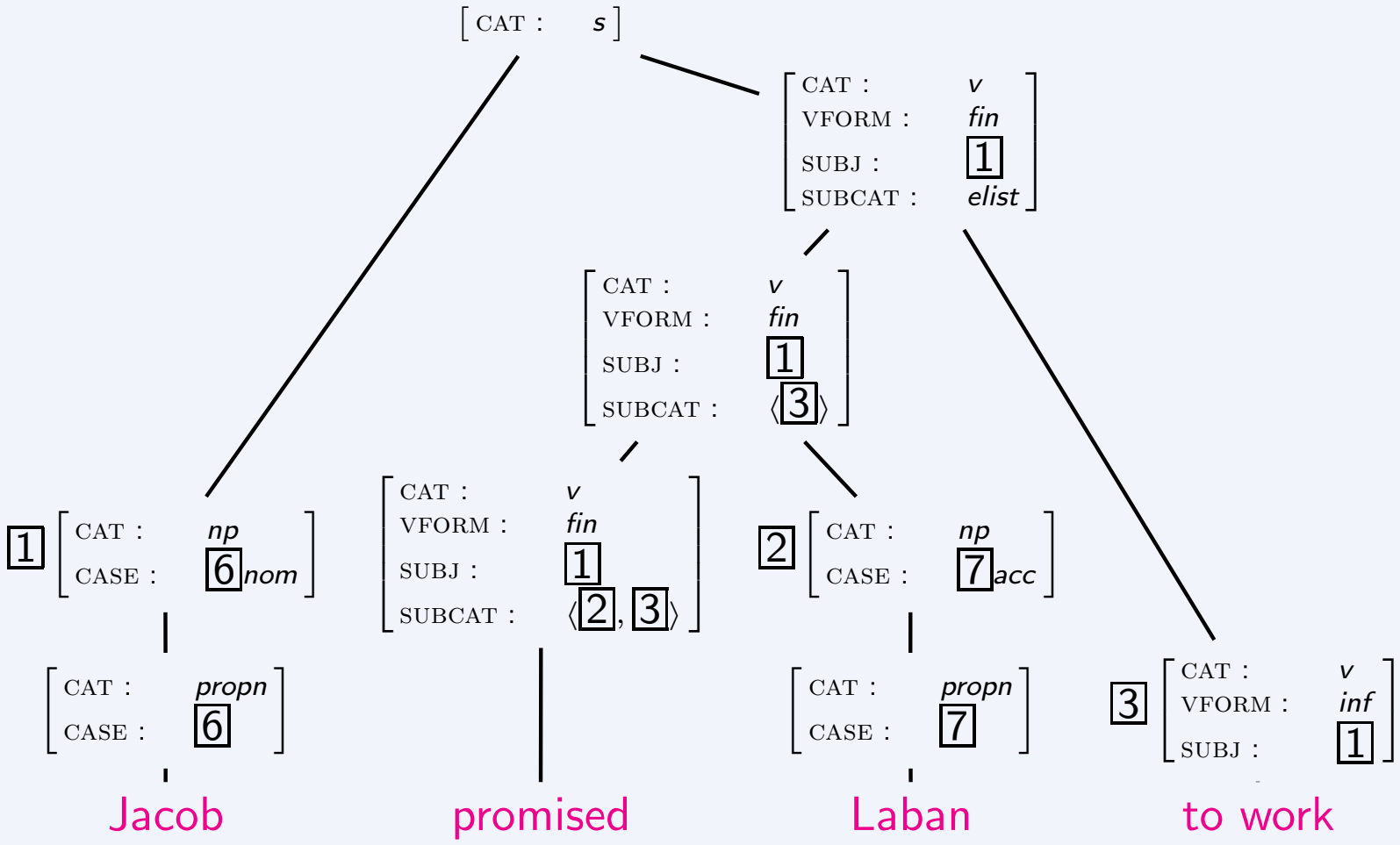
The lexical entry of promise

Example (The lexical entry of promise)

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

A derivation tree for Jacob promised Laban to work

Example (A derivation tree for Jacob promised Laban to work)



The lexical entry of persuade

Example (The lexical entry of persuade)

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

Constituent coordination

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].

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]!"

Coordination in CFG

Example (Coordination in CFG)

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

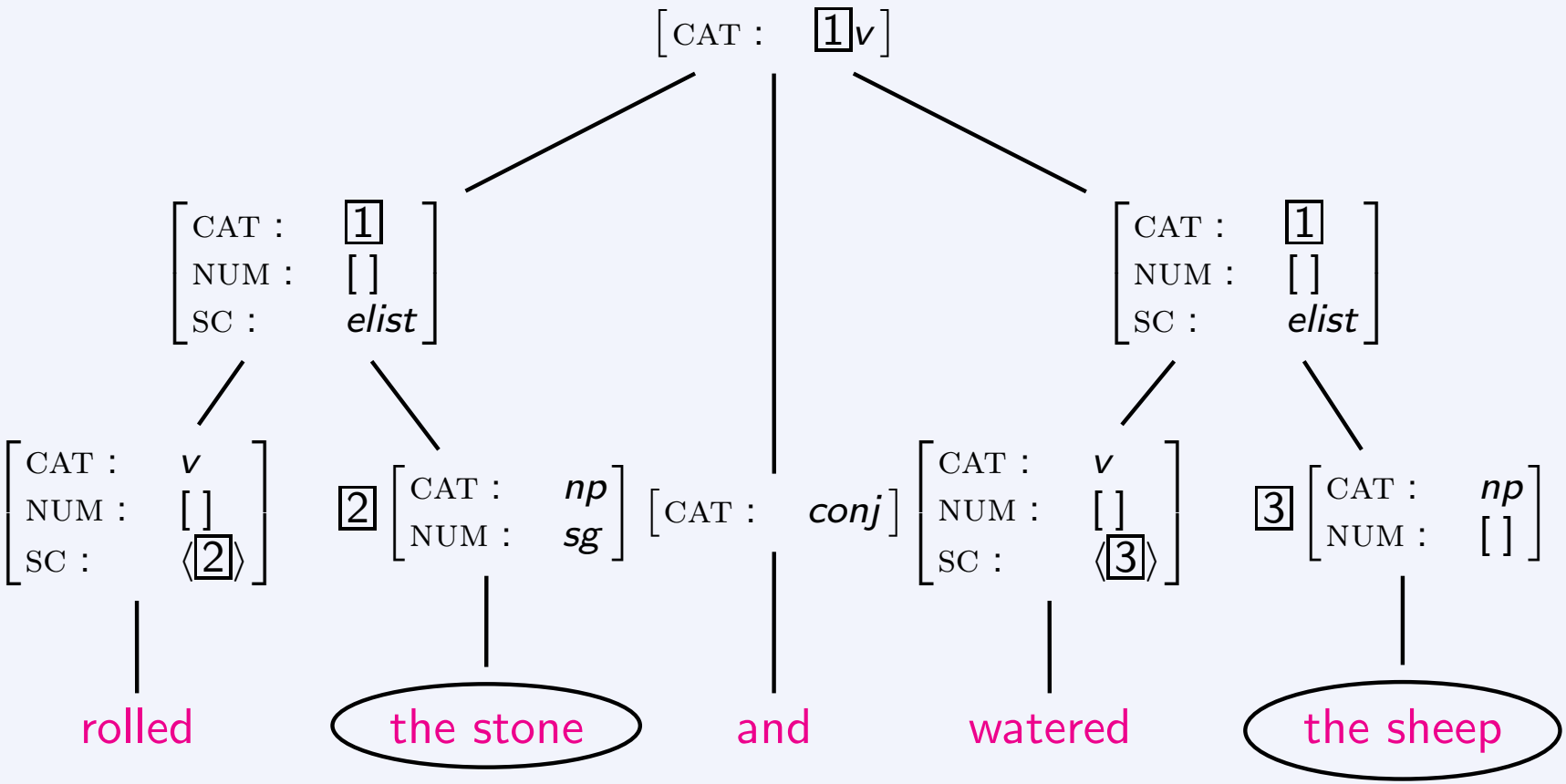
Coordination in UG

Example (Coordination in UG)

$$\begin{bmatrix}\text{CAT}: & \boxed{1}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & \boxed{1}\end{bmatrix} \begin{bmatrix}\text{CAT}: & \textit{conj}\end{bmatrix} \begin{bmatrix}\text{CAT}: & \boxed{1}\end{bmatrix}$$

Coordination in UG

Example (Coordination)



Tough issues in coordination

- Coordination of conjunctions
- Properties of the conjoined phrases
- Coordination of unlikes
- Non-constituent coordination

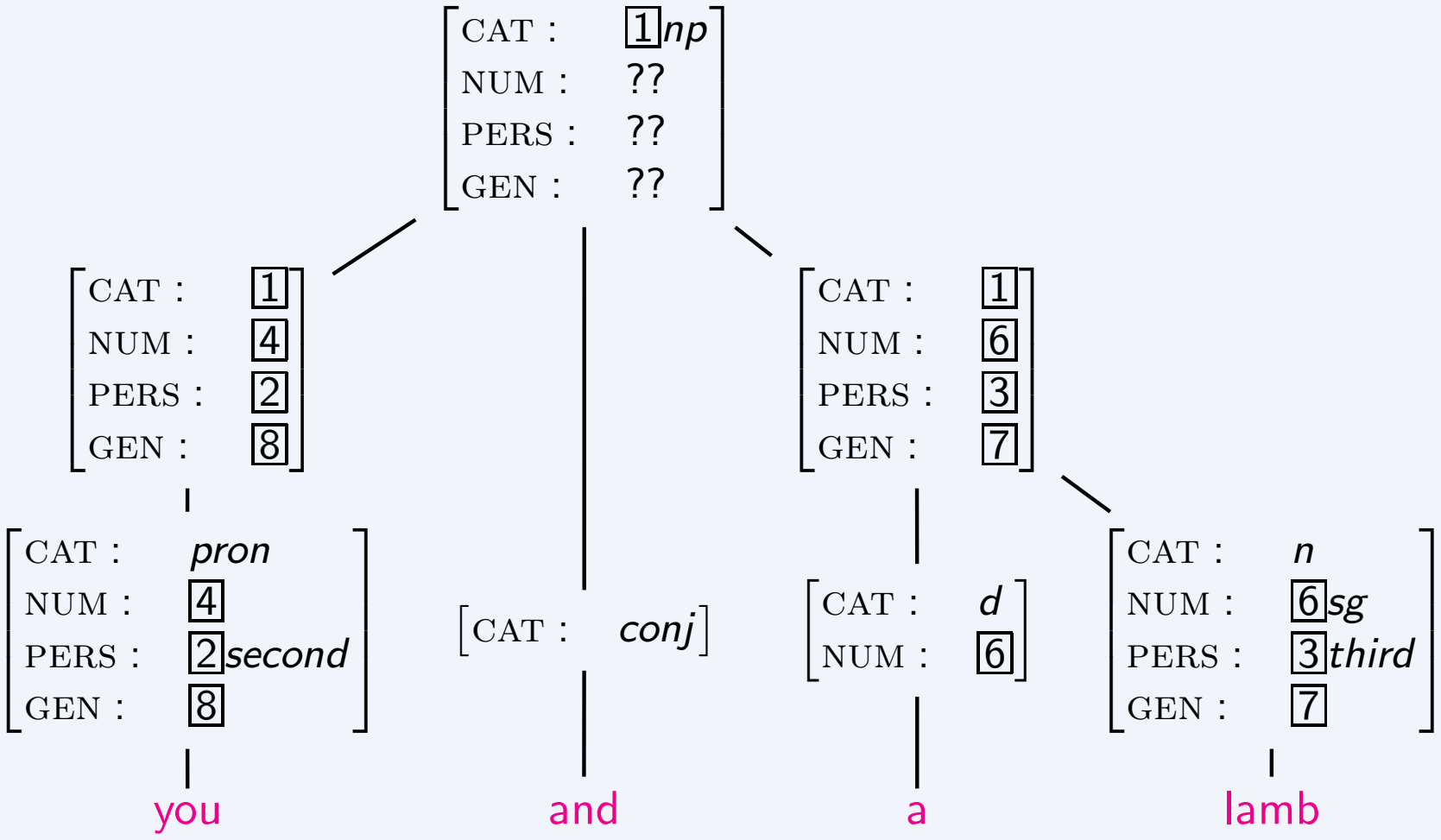
Coordination

Example (Ruling out coordination in UG)

$$\begin{bmatrix} \text{CAT}: & \boxed{1} \\ \text{CONJ}: & - \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & \boxed{1} \\ \text{CONJ}: & + \end{bmatrix} \begin{bmatrix} \text{CAT}: & \textit{conj} \end{bmatrix} \begin{bmatrix} \text{CAT}: & \boxed{1} \\ \text{CONJ}: & + \end{bmatrix}$$

Coordination

Example (Properties of the conjoined phrases)



Coordination

Example (Coordination of unlikes)

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

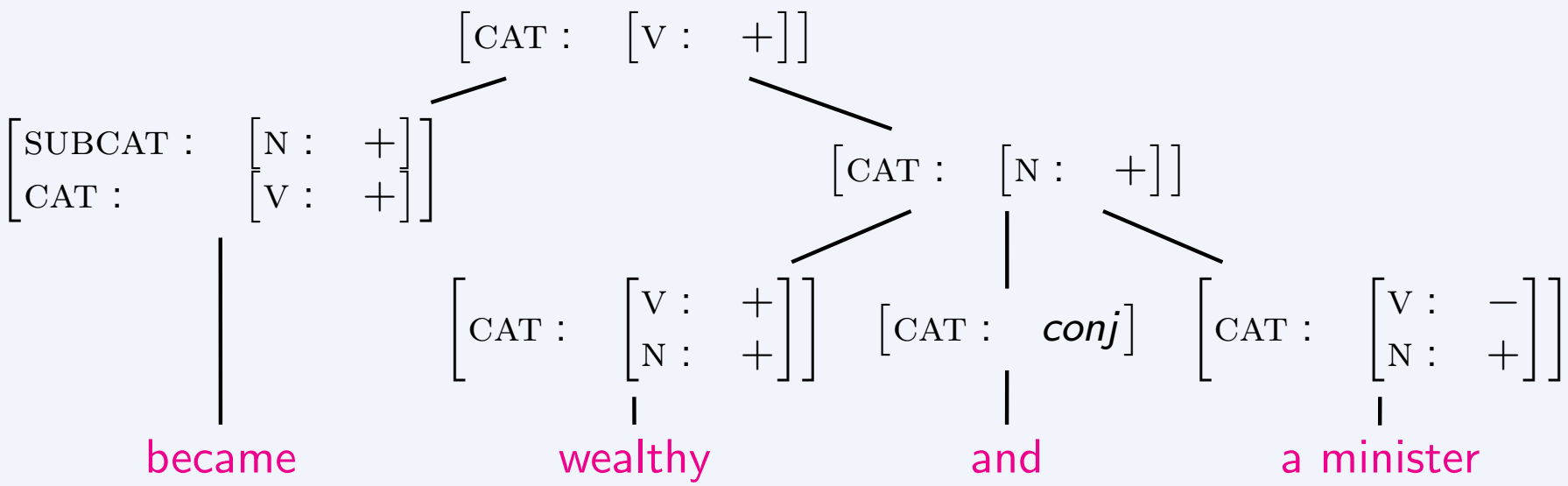
Example (Coordination of unlikes)

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

where '$\sqcap$' is the generalization operator

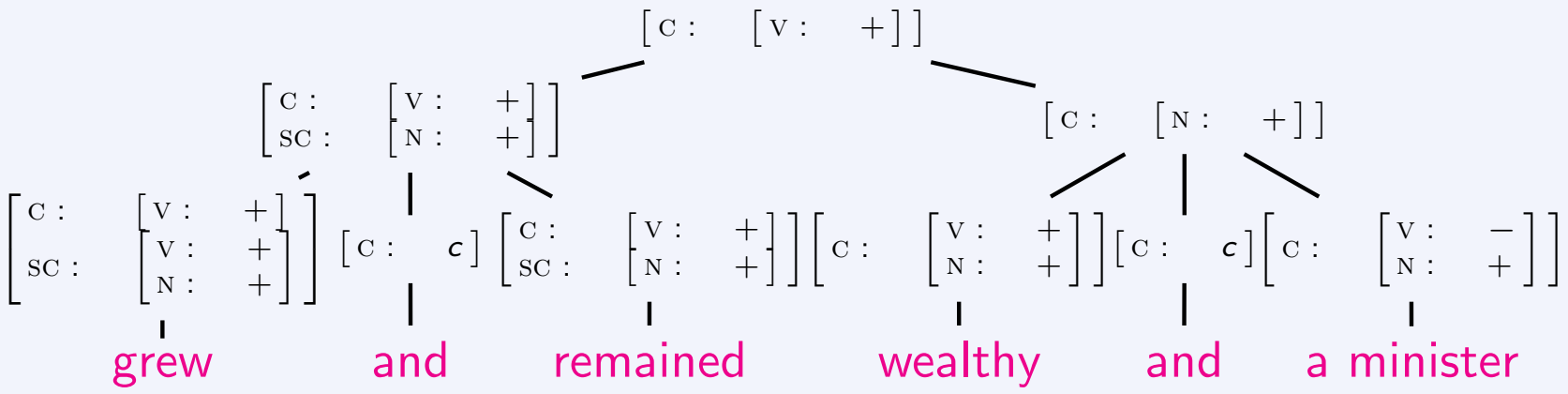
Coordination

Example (Coordination of unlikes)



Coordination

Example (Coordination of unlikes)



Coordination

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

Unification grammars facilitate linguistic generalizations

- Compared with context-free grammars, unification grammars provide much better means for expressing linguistic generalizations
  - Verb subcategorization
  - Coordination
- Unification grammars also provide much more informative structures than CFGs
  - Agreement
  - Subject/object control
- Unification grammars provide a very powerful tool for expressing what other linguistic theories would call "movement"
  - Gap–filler constructions
  - Unbounded dependencies

Expressiveness of unification grammars

- We hinted above that unification grammars are more expressive than CFGs
- Unification grammars are strictly more powerful than CFGs, even when *weak* generation capacity is concerned
- We show two unification grammars for formal languages that are known to be trans-context-free

Unification grammars are more expressive than CFGs

- G_{abc} generates the language $L = \{a^n b^n c^n \mid n > 0\}$
- 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

Unification grammars are more expressive than CFGs

- Feature structures in this example 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 AFS (that is, an AMRS of length 1) whose depth is *n*
- For example, the string *bbb* is derived by the following feature structure:

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

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

Example (A unification grammar G_{abc} for $\{a^n b^n c^n \mid n > 0\}$)

$$\rho_1 : \begin{bmatrix} \text{CAT} : & s \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : & ap \\ \text{T} : & \boxed{3} \end{bmatrix} \begin{bmatrix} \text{CAT} : & bp \\ \text{T} : & \boxed{3} \end{bmatrix} \begin{bmatrix} \text{CAT} : & cp \\ \text{T} : & \boxed{3} \end{bmatrix}$$

$$\rho_2 : \begin{bmatrix} \text{CAT} : & ap \\ \text{T} : & \begin{bmatrix} \text{T} : & \boxed{4} \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : & at \end{bmatrix} \begin{bmatrix} \text{CAT} : & ap \\ \text{T} : & \boxed{4} \end{bmatrix}$$

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

$$\rho_4 : \begin{bmatrix} \text{CAT} : & bp \\ \text{T} : & \begin{bmatrix} \text{T} : & \boxed{4} \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : & bt \end{bmatrix} \begin{bmatrix} \text{CAT} : & bp \\ \text{T} : & \boxed{4} \end{bmatrix}$$

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

Example (A unification grammar G_{abc} for $\{a^n b^n c^n \mid n > 0\}$)

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

$$\rho_6 : \begin{bmatrix} \text{CAT} : & cp \\ \text{T} : & \begin{bmatrix} \text{T} : & \boxed{4} \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT} : & ct \end{bmatrix} \begin{bmatrix} \text{CAT} : & cp \\ \text{T} : & \boxed{4} \end{bmatrix}$$

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

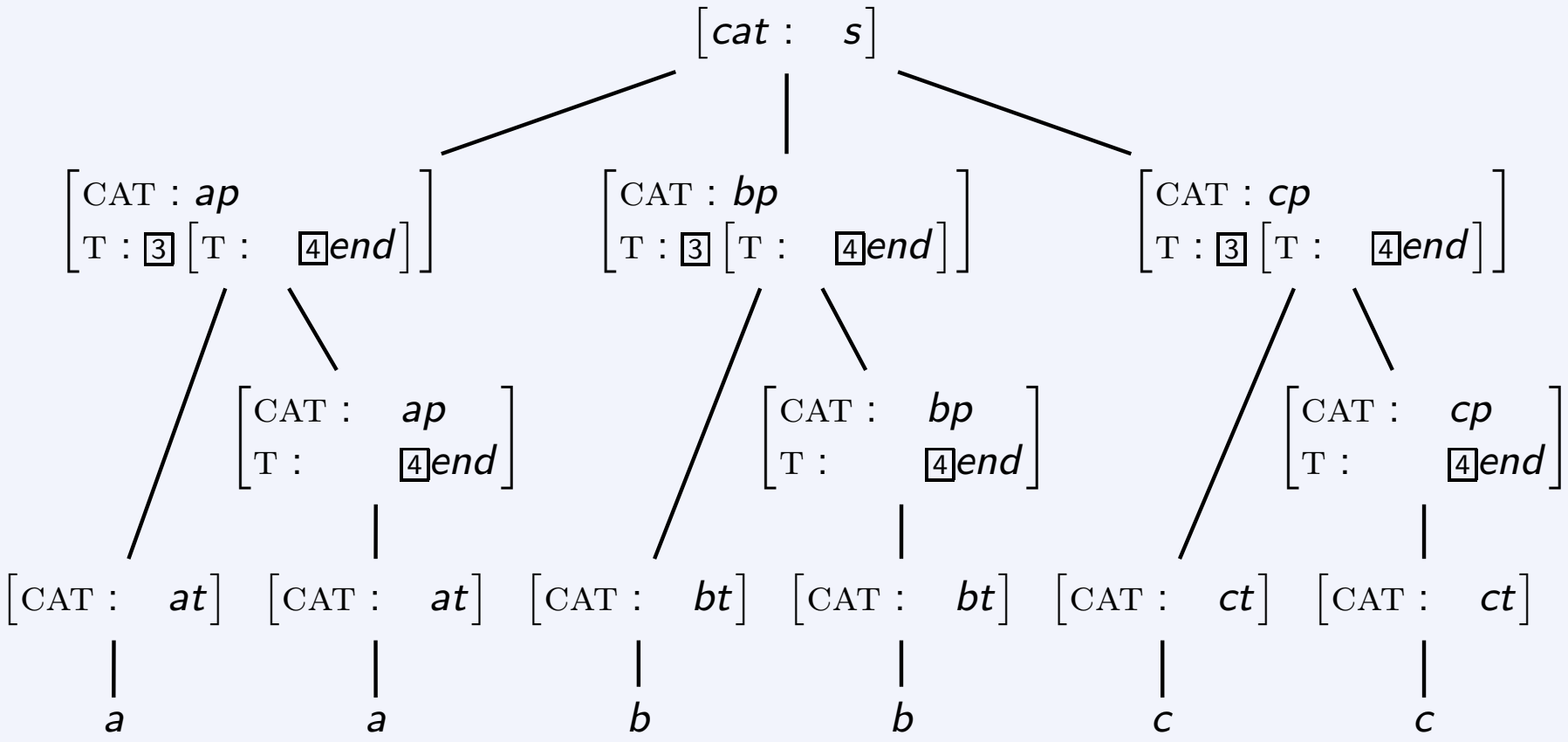
$$\begin{bmatrix} \text{CAT} : & at \end{bmatrix} \rightarrow a$$

$$\begin{bmatrix} \text{CAT} : & bt \end{bmatrix} \rightarrow b$$

$$\begin{bmatrix} \text{CAT} : & ct \end{bmatrix} \rightarrow c$$

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

Example (Derivation tree of $a^2 b^2 c^2$)



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

Corollary

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

A unification grammar G_{ww} for $\{ww \mid w \in \{a,b\}^+\}$

Example (A unification grammar G_{ww} for $\{ww \mid w \in \{a,b\}^+\}$)

$$\begin{bmatrix}\text{CAT}: & s\end{bmatrix} \rightarrow \begin{bmatrix}\text{FIRST}: & \boxed{4} \\ \text{REST}: & \boxed{2}\end{bmatrix} \begin{bmatrix}\text{FIRST}: & \boxed{4} \\ \text{REST}: & \boxed{2}\end{bmatrix}$$

$$\begin{bmatrix}\text{FIRST}: & ap \\ \text{REST}: & \begin{bmatrix}\text{FIRST}: & \boxed{4} \\ \text{REST}: & \boxed{2}\end{bmatrix}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & at\end{bmatrix} \begin{bmatrix}\text{FIRST}: & \boxed{4} \\ \text{REST}: & \boxed{2}\end{bmatrix}$$

$$\begin{bmatrix}\text{FIRST}: & bp \\ \text{REST}: & \begin{bmatrix}\text{FIRST}: & \boxed{4} \\ \text{REST}: & \boxed{2}\end{bmatrix}\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & bt\end{bmatrix} \begin{bmatrix}\text{FIRST}: & \boxed{4} \\ \text{REST}: & \boxed{2}\end{bmatrix}$$

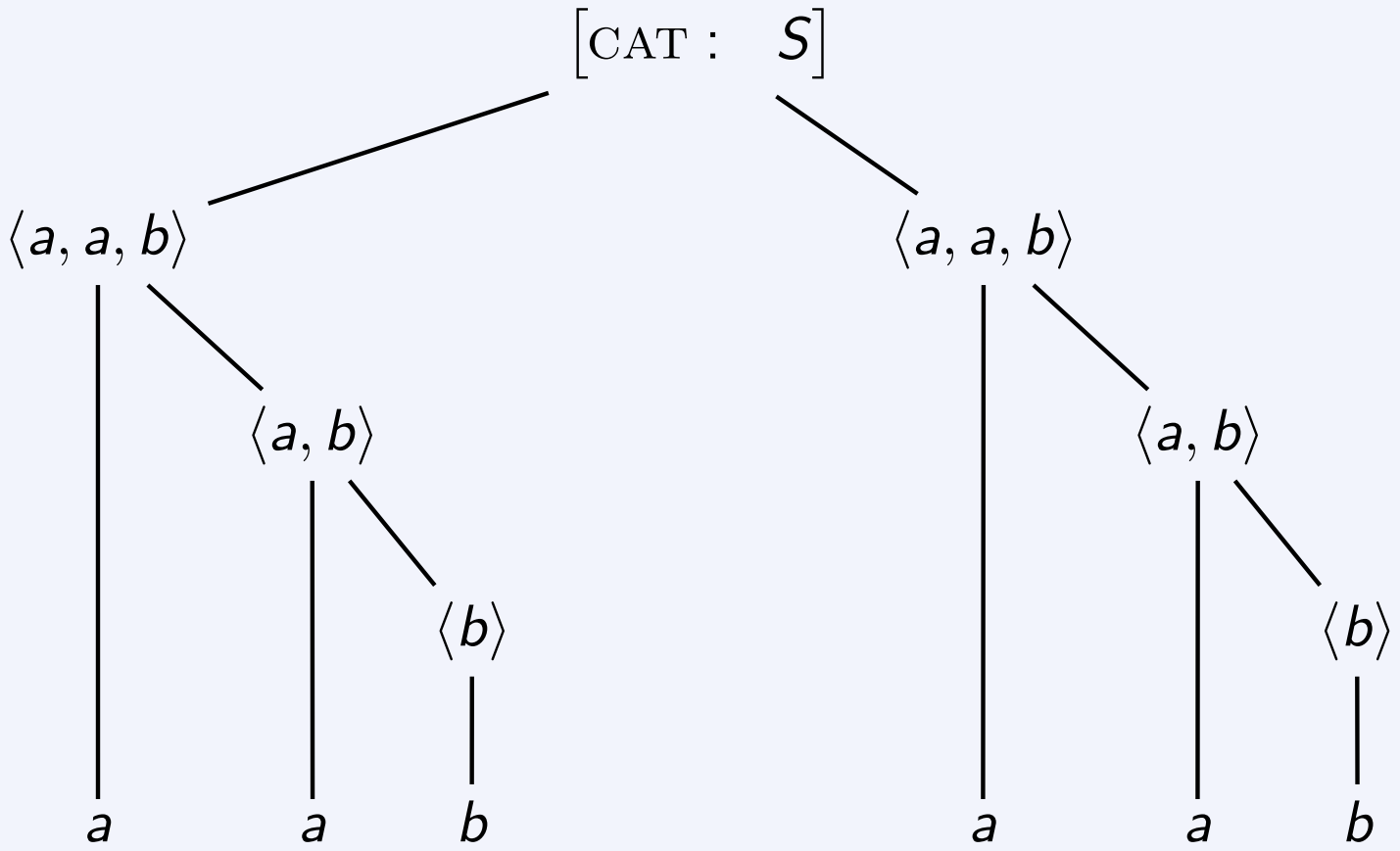
$$\begin{bmatrix}\text{FIRST}: & ap \\ \text{REST}: & elist\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & at\end{bmatrix}$$

$$\begin{bmatrix}\text{FIRST}: & bp \\ \text{REST}: & elist\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & bt\end{bmatrix}$$

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

A unification grammar G_{ww} for $\{ww \mid w \in \{a, b\}^+\}$

Example (A derivation tree for the string *aabaab*)



Unification grammars and Turing machines

- How expressive are unification grammars?
- They are equivalent in their weak generative power to *unrestricted rewriting systems*
- Unification grammars are equivalent to Turing machines in their generative capacity
- The languages generated by unification grammars are exactly the set of *recursively enumerable* languages
- The universal recognition problem with unification grammars is undecidable: given an arbitrary unification grammar G and a string w, no computational procedure can be designed to determine whether $w \in L(G)$

Turing machines

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

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 $\sigma \in \Sigma$
- 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.

Turing machines

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

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

Turing machines

- where:

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

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

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

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

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
- 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:
  - $\textsc{Feats} = \{\textsc{cat}, \textsc{left}, \textsc{right}, \textsc{curr}, \textsc{first}, \textsc{rest}\}$
  - $\textsc{Atoms} = \Sigma \cup \{\textit{start}, \textit{elist}\}$
  - The start symbol is $\begin{bmatrix}\textsc{cat} : & \textit{start}\end{bmatrix}$
  - the only terminal symbol is halt

Unification grammars and Turing machines: rules

- Two rules are defined for every Turing machine:

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

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

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:

$$\begin{bmatrix} \text{CAT}: & q \\ \text{CURR}: & \sigma \\ \text{RIGHT}: & \boxed{4} \\ \text{LEFT}: & \boxed{2} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{CURR}: & \sigma' \\ \text{RIGHT}: & \boxed{4} \\ \text{LEFT}: & \boxed{2} \end{bmatrix}$$

Unification grammars and Turing machines: rules

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

$$\begin{bmatrix} \text{CAT}: & q \\ \text{CURR}: & \sigma \\ \text{RIGHT}: & \textit{elist} \\ \text{LEFT}: & \boxed{4} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{CURR}: & \flat \\ \text{RIGHT}: & \textit{elist} \\ \text{LEFT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & \boxed{4} \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & q \\ \text{CURR}: & \sigma \\ \text{RIGHT}: & \begin{bmatrix} \text{FIRST}: & \boxed{4} \\ \text{REST}: & \boxed{2} \end{bmatrix} \\ \text{LEFT}: & \boxed{5} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{CURR}: & \boxed{4} \\ \text{RIGHT}: & \boxed{2} \\ \text{LEFT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & \boxed{5} \end{bmatrix} \end{bmatrix}$$

Unification grammars and Turing machines: rules

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

$$\begin{bmatrix} \text{CAT}: & q \\ \text{CURR}: & \sigma \\ \text{RIGHT}: & \boxed{4} \\ \text{LEFT}: & \mathit{elist} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{CURR}: & \flat \\ \text{RIGHT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & \boxed{4} \end{bmatrix} \\ \text{LEFT}: & \mathit{elist} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & q \\ \text{CURR}: & \sigma \\ \text{RIGHT}: & \boxed{4} \\ \text{LEFT}: & \begin{bmatrix} \text{FIRST}: & \boxed{2} \\ \text{REST}: & \boxed{5} \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & p \\ \text{CURR}: & \boxed{2} \\ \text{RIGHT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & \boxed{4} \end{bmatrix} \\ \text{LEFT}: & \boxed{5} \end{bmatrix}$$

Unification grammars and Turing machines: results

Lemma

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.

Theorem

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

Corollary

The universal recognition problem for unification grammars is undecidable.

Unification grammars and Turing machines

- Unification grammars are indeed a model of computation: every recursively enumerable set can be computed as the language generated by some unification grammar
- Consider again the simulation of a Turing machine M by a unification grammar G_M
- Feature structures manipulated by the grammar encode the contents of the Turing machine tape
- By the end of the derivation, the pre-terminal of the terminal halt is a feature structure which encodes, in its RIGHT and LEFT features, the contents of the tape when the Turing machine halts

Unification grammars and Turing machines

- $w \in L(M)$ iff there exists a terminating computation of M where w is the contents of the tape
- It is therefore possible to define, for each Turing machine M, a grammar G'_M, such that $L(G'_M) = L(M)$, in the following way
- First, G'_M is constructed in a similar way to G_M, simulating the operation of M until it terminates (or indefinitely, in case it does not terminate)
- Then, additional rules distinguish G'_M from G_M
- Such rules should first copy the contents of the LEFT feature to the beginning of the RIGHT list
- Then, additional rules should pop the contents of the RIGHT list, one by one, and generate a pre-terminal for each of the list's elements

Unification grammars and Turing machines

Example (Unification grammars and Turing machines)

$$h \rightarrow \begin{bmatrix} \text{CAT}: & \textit{shift} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & \textit{shift} \\ \text{RIGHT}: & \boxed{4} \\ \text{LEFT}: & \textit{elist} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & \textit{print} \\ \text{RIGHT}: & \boxed{4} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & \textit{shift} \\ \text{RIGHT}: & \boxed{2} \\ \text{LEFT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & \boxed{4} \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & \textit{shift} \\ \text{RIGHT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & \boxed{2} \end{bmatrix} \\ \text{LEFT}: & \boxed{4} \end{bmatrix}$$

$$\begin{bmatrix} \text{CAT}: & \textit{print} \\ \text{RIGHT}: & \textit{elist} \end{bmatrix} \rightarrow \epsilon$$

$$\begin{bmatrix} \text{CAT}: & \textit{print} \\ \text{RIGHT}: & \begin{bmatrix} \text{FIRST}: & \sigma \\ \text{REST}: & \boxed{4} \end{bmatrix} \end{bmatrix} \rightarrow \sigma \begin{bmatrix} \text{CAT}: & \textit{print} \\ \text{RIGHT}: & \boxed{4} \end{bmatrix}$$

Extensions and open problems

- Restricted versions of unification grammars
  - Off-line parsability
  - Context-free and Mildly-context-sensitive unification grammars
  - Polynomially-arsable unification grammars
- Typed unification grammars
  - Type hierarchies
  - Appropriateness specification
  - Type inference
- Development of large-scale grammars
  - Grammar engineering
  - Modularity, information encapsulation, separate compilation, ...