Internalizing categories

- The grammars we have seen so far had an explicit *context-free backbone* (or *skeleton*), obtained by considering the (context-free) grammar induced by the base categories.
- This is not imposed by the formalism; rather, the base categories can be *internalized* into the feature structures themselves.

Internalizing categories

For example, the rule

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

can be re-written as

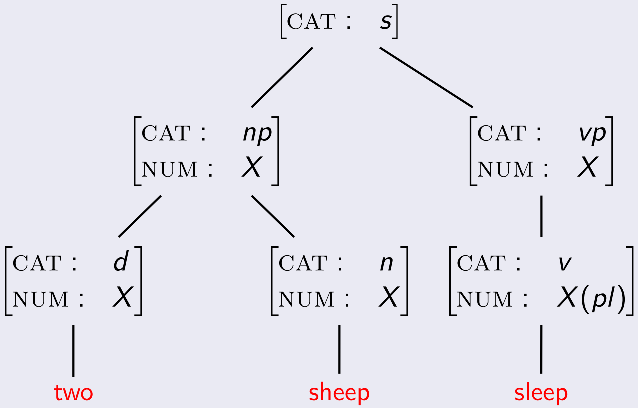
$$\begin{bmatrix}\text{CAT}: & np \\ \text{NUM}: & X\end{bmatrix} \rightarrow \begin{bmatrix}\text{CAT}: & d \\ \text{NUM}: & X\end{bmatrix} \begin{bmatrix}\text{CAT}: & n \\ \text{NUM}: & X\end{bmatrix}$$

Internalizing categories

- In the new presentation of grammars, productions are essentially multi-AVMs.
- Derivations, derivation trees, languages...
- Special features and the signature.

Internalizing categories

Example: Derivation tree



Internalizing categories

- Once the base category of a phrase is admitted as the value of one of the features in the feature structure associated with that phrase, it does not have to be represented as an *atomic* value.
- For example, the Chomskian representation of categories:

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

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

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

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

Internalizing categories

- *Internalization* of the category results in additional expressive power.
- It now becomes possible to consider feature structures in which the value of the CAT feature is underspecified, or even unrestricted.
- For example, one might describe *a phrase in singular* using the feature structure

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

Internalizing categories

- Once information about the category of a phrase is embedded within the feature structure, it can be manipulated in more ways than simply encoding the category of a phrase.
- Internalized categories will be used to:
  - represent information about the subcategories of verbs
  - list information about constituents that are "moved", or "transformed", using the *slash* notation
  - account for coordination.

Subcategorization lists

- Motivation: to account for the subcategorization data in a more general, elegant way, extending the coverage of our grammar from the smallest fragment E_0 to the fragment E_1.
- In E_1 different verbs *subcategorize for* different kinds of complements: noun phrases, infinitival verb phrases, sentences etc. Also, some verbs require more than one complement.
- The idea behind the solution is to store in the lexical entry of each verb not an atomic feature indicating its subcategory, but rather a *list* of atomic categories, indicating the appropriate complements of the verb.

Subcategorization lists

Example: Lexical entries of verbs using subcategorization lists

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

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

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

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

Subcategorization lists

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

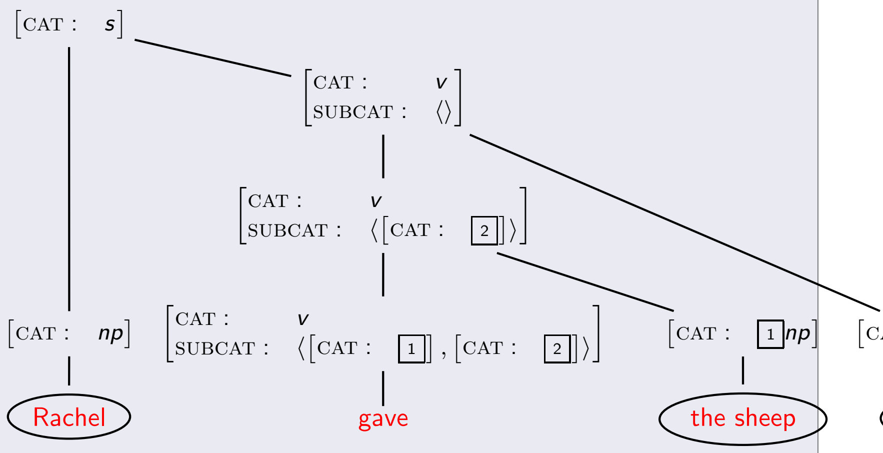
Example: VP rules using subcategorization lists

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

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

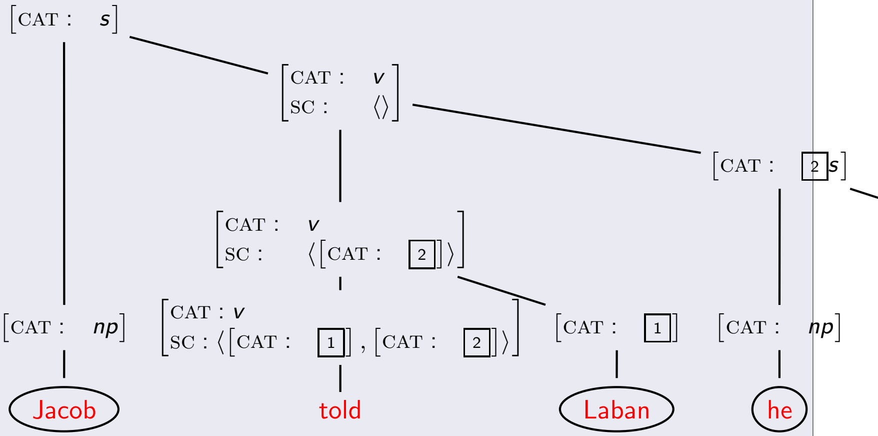
Subcategorization lists

Example: A derivation tree



Subcategorization lists

Example: A derivation tree



Subcategorization lists

- In the above grammar, categories on subcategorization lists are represented as an atomic symbol.
- The method outlined here can be used with more complex encodings of categories. In other words, the specification of categories in a subcategorization list can include all the constraints that the verb imposes on its complements

Subcategorization lists

Example: Subcategorization imposes case constraints

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

Subcategorization lists

The lexical entry of gebe, then, could be:

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

The VP rule has to be slightly modified:

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

G_3, a complete E_1-grammar

Example: G_3, a complete E_1-grammar

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

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

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

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

Example: (continued)

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

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

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

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

Example: (continued)

$$\text{lamb} \longrightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & sg \\ \text{CASE}: & Y \end{bmatrix} \qquad \text{lambs} \longrightarrow \begin{bmatrix} \text{CAT}: & n \\ \text{NUM}: & pl \\ \text{CASE}: & Y \end{bmatrix}$$

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

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

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

Long distance dependencies

- Internalized categories are very useful in the treatment of unbounded dependencies, which are included in the grammar fragment E_3.
- Such phenomena involve a "missing" constituent that is realized outside the clause from which it is missing, as in:

  (1) The shepherd wondered whom Jacob loved $\smile$.
  (2) The shepherd wondered whom Laban thought Jacob loved $\smile$.
  (3) The shepherd wondered whom Laban thought Rachel claimed Jacob loved $\smile$.

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

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

Long distance dependencies

- The gap need not be in the object position:

  (5) Jacob wondered who $\smile$ *loved Leah*
  (6) Jacob wondered who Laban believed $\smile$ *loved Leah*

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

  (7) Jacob wondered who the shepherd loved Leah

Long distance dependencies

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

  (8a) This is the well which Jacob is likely to ⌣ draw water from ⌣
  (8b) It was Leah that Jacob worked for ⌣ without loving ⌣

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

Long distance dependencies

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

  (9) Rachel, Jacob loved $\smile$
  (10) Rachel, every shepherd knew Jacob loved $\smile$

- Another example is *interrogative sentences*:

  (11) Who did Jacob love $\smile$*?*
  (12) Who did Laban believe Jacob loved $\smile$*?*

  We do not account for such phenomena here.

Long distance dependencies

- Phrases such as whom Jacob loved $\smile$ or who $\smile$ loved Rachel are instances of a category that we haven't discussed yet.
- They are basically *sentences*, with a constituent which is "moved" from its default position and realized as a wh-pronoun in front of the phrase.
- We will represent such phrases by using the same category, *s*, which we used for sentences; but to distinguish them from declarative sentences we will add a feature, QUE, to the category. The value of QUE will be '+' in sentences with an interrogative pronoun realizing a transposed constituent.

Long distance dependencies

We add a lexical entry for the pronoun whom:

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

and update the rule that derives pronouns:

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

Long distance dependencies

- We now propose an extension of G_3 that can handle long distance dependencies.
- The idea is to allow partial phrases, such as Jacob loved $\smile$, to be derived from a category that is similar to the category of the full phrase, in this case Jacob loved Rachel; but to signal in some way that a constituent, in this case a noun phrase, is missing.
- We extend G_3 with two additional rules, based on the first two rules of G_3.

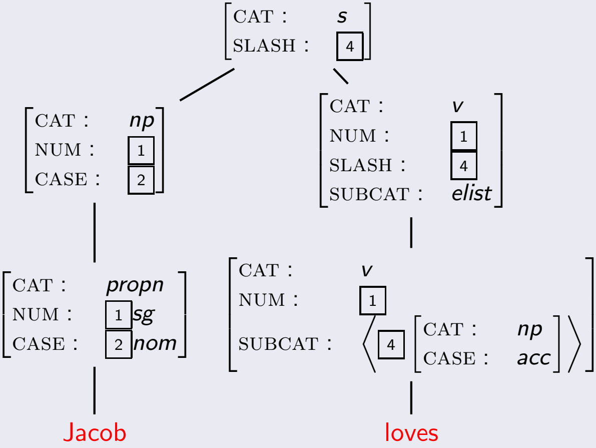
Long distance dependencies

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

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

Long distance dependencies

Example: A derivation tree for Jacob loves



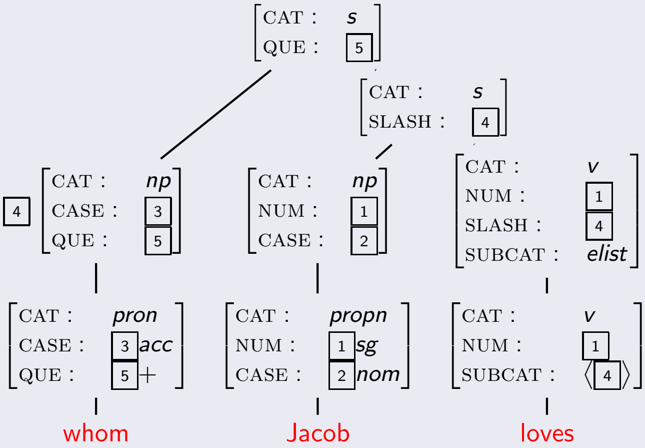
Long distance dependencies

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

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

Long distance dependencies

Example: A derivation tree for whom Jacob loves ◡



Long distance dependencies

- Unbounded dependencies can hold across several clause boundaries:

  The shepherd wondered whom Jacob loved $\smile$.
  The shepherd wondered whom Laban thought Jacob loved $\smile$.
  The shepherd wondered whom Laban thought Leah claimed Jacob loved $\smile$.

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

  The shepherd wondered who $\smile$ loved Rachel.
  The shepherd wondered who Laban thought $\smile$ loved Rachel.
  The shepherd wondered who Laban thought Leah claimed $\smile$ loved Rachel.

Long distance dependencies

- The solution we proposed for the simple case of unbounded dependencies can be easily extended to the more complex examples:
  - a slash introduction rule;
  - slash propagation rules;
  - and a gap filler rule.
- In order to account for filler-gap relations that hold across several clauses, all that needs to be done is to add more slash propagation rules.

Long distance dependencies

- For example, in

  > *The shepherd wondered whom Laban thought Jacob loved* $\smile$.

  the slash is introduced by the verb phrase loved $\smile$, and is propagated to the sentence Jacob loved $\smile$ by rule (3).
- A rule that propagates the value of SLASH from a sentential object to the verb phrase of which it is an object:

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

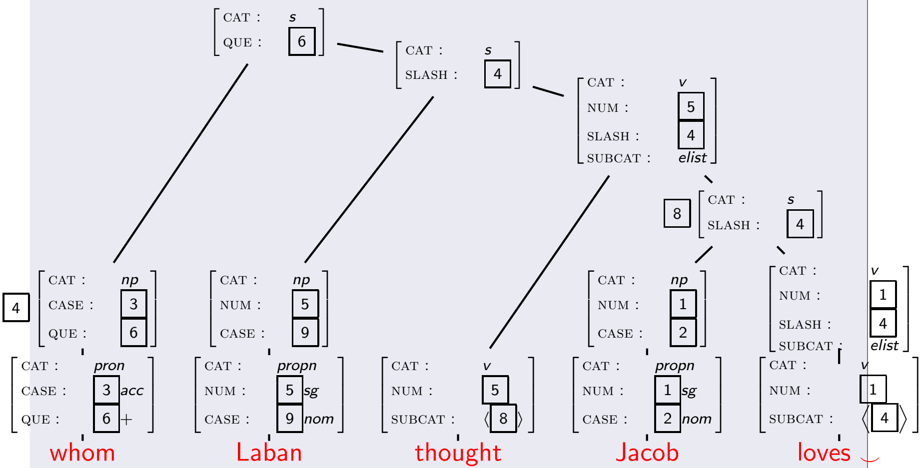
Long distance dependencies

- Then, the slash is propagated from the verb phrase thought Jacob loved $\smile$ to the sentence Laban thought Jacob loved $\smile$:

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

Long distance dependencies

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



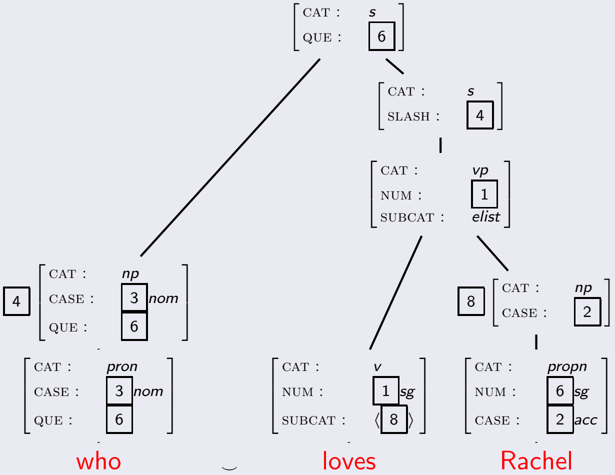
Long distance dependencies

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

$$(8)\quad \begin{bmatrix} \textsc{cat}: & s \\ \textsc{slash}: & \begin{bmatrix} \textsc{cat}: & np \\ \textsc{num}: & X \\ \textsc{case}: & nom \end{bmatrix} \end{bmatrix} \rightarrow \begin{bmatrix} \textsc{cat}: & v \\ \textsc{num}: & X \\ \textsc{subcat}: & elist \end{bmatrix}$$

Long distance dependencies

Example: A derivation tree for who ⌣ loves Rachel



Subject and object control

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

  Jacob promised Laban to work seven years
  Laban persuaded Jacob to work seven years

- The key observation in the solution is that the differences between the two examples stem from differences in the matrix verbs:
  - promise is a *subject control* verb
  - persuade is *object control*.

Subject and object control

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

Example: G_4: explicit SUBJ values

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

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

Subject and object control

Example: (continued)

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

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

Subject and object control

Example: (continued)

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

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

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

Example: (continued)

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

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

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

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

Subject and object control

Accounting for infinitival verb phrases:

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

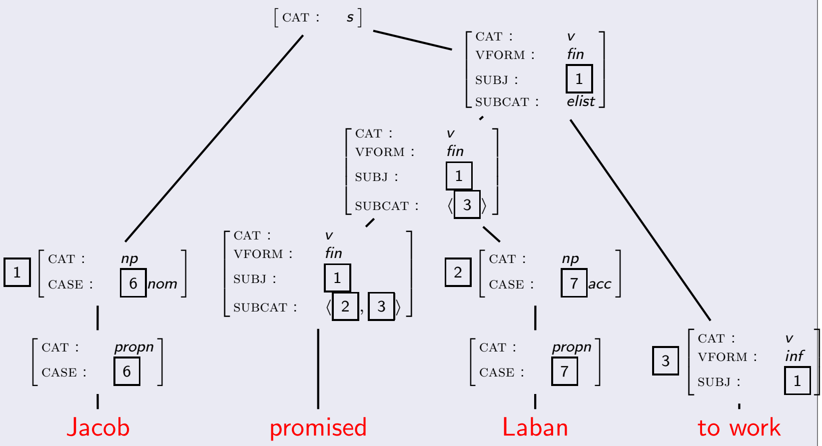
Subject and object control

The lexical entries of verbs such as promise or persuade:

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

Subject and object control

Example: A derivation tree for Jacob promised Laban to work



Subject and object control

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

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

Constituent coordination

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

Constituent coordination

Example:

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]!"

Constituent coordination

- We extend the grammar fragment to cover coordination, referring to it as E_4.
- The lexicon of a grammar for E_4 is extended by a closed class of *conjunction* words; categorized under *Conj*, this class includes the words and, or, but and perhaps a few others (E_4 contains only these three).
- We assume that in E_4, *every* category of E_0 can be conjoined.
- We also assume – simplifying a little – that the same conjunctions are possible for all the categories.

Constituent coordination

- A context-free grammar for coordination:

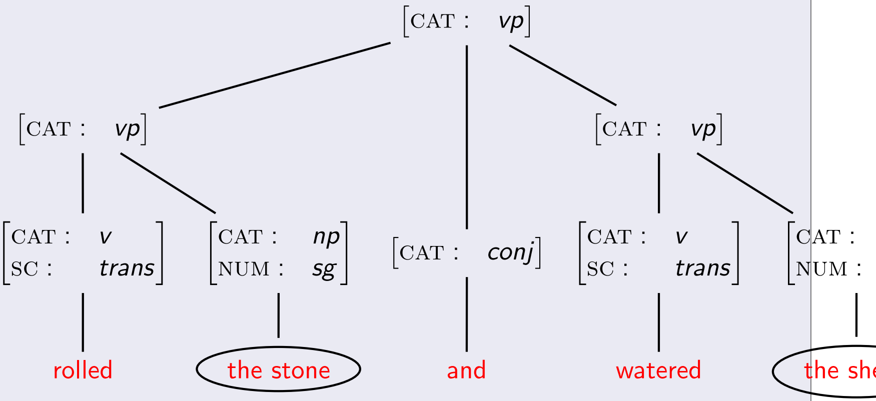
$$\begin{array}{ccl} S & \rightarrow & S\ Conj\ S \\ NP & \rightarrow & NP\ Conj\ NP \\ VP & \rightarrow & VP\ Conj\ VP \\ \vdots & & \\ Conj & \rightarrow & \text{and, or, but, } \ldots \end{array}$$

- With generalized categories, a single production is sufficient:

$$\begin{bmatrix}\textsc{cat}: & X\end{bmatrix} \rightarrow \begin{bmatrix}\textsc{cat}: & X\end{bmatrix} \begin{bmatrix}\textsc{cat}: & conj\end{bmatrix} \begin{bmatrix}\textsc{cat}: & X\end{bmatrix}$$

Constituent coordination

Example: Coordination



Constituent coordination

The above solution is over-simplifying:

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

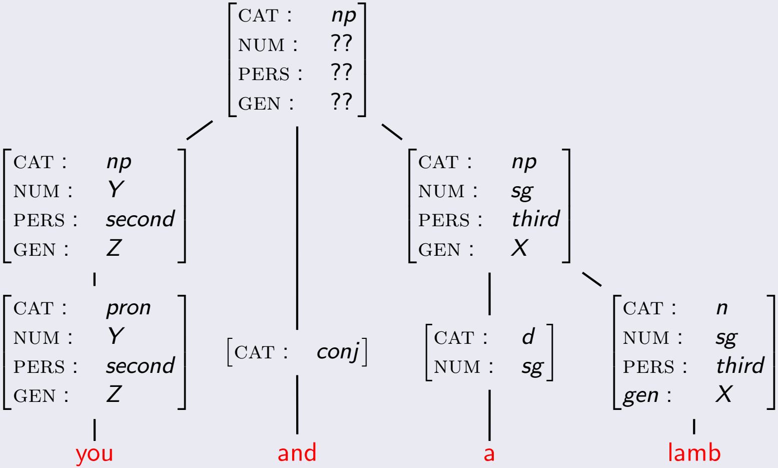
Constituent coordination

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

$$\begin{bmatrix} \text{CAT}: & X \\ \text{CONJ'BLE}: & - \end{bmatrix} \rightarrow \begin{bmatrix} \text{CAT}: & X \\ \text{CONJ'BLE}: & + \end{bmatrix} \begin{bmatrix} \text{CAT}: & \mathit{conj} \end{bmatrix} \begin{bmatrix} \text{CAT}: & X \\ \text{CONJ'BLE}: & + \end{bmatrix}$$

Properties of conjoined constituents

Example: NP coordination



Coordination of unlikes

- Consider the following English data:

  Joseph became wealthy
  Joseph became a minister
  Joseph became [wealthy and a minister]
  Joseph grew wealthy
  $*$*Joseph grew a minister*
  $*$*Joseph grew [wealthy and a minister]*

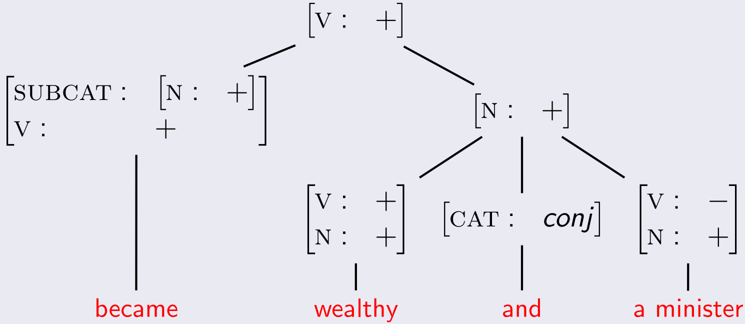
- These data are easy to account for in a unification-based framework with a possibility of specifying generalization instead of unification in certain cases:

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

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

Coordination of unlikes

Example:



Coordination of unlikes

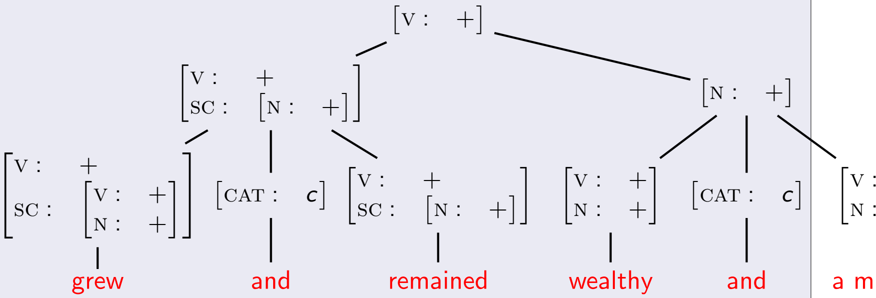
- The situation becomes more complicated when *verbs*, too, are conjoined:

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

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

Coordination of unlikes

Example:



Non-constituent coordination

Sometimes *non*-constituents can be conjoined:

Rachel gave the sheep [grass] and [water]
Rachel gave [the sheep grass] and [the lambs water]
Rachel [kissed] and Jacob [hugged] Binyamin

Expressiveness of unification grammars

- Just how expressive are unification grammars?
- What is the class of languages generated by unification grammars?

Trans-context-free languages

- A grammar, G_{abc}, for the language $L = \{a^n b^n c^n \mid n > 0\}$.
- Feature structures will have two features: CAT, which stands for category, and T, which "counts" the length of sequences of a-s, b-s and c-s.
- The "category" is *ap* for strings of a-s, *bp* for b-s and *cp* for c-s. The categories *at*, *bt* and *ct* are pre-terminal categories of the words *a*, *b* and *c*, respectively.
- "Counting" is done in unary base: a string of length n is derived by an AVM (that is, an multi-AVM of length 1) whose depth is n.
- For example, the string *bbb* is derived by the following AVM:

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

Trans-context-free languages

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

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

$$\rho_1 : \quad \begin{bmatrix} \textsc{cat}: & s \end{bmatrix} \;\longrightarrow\; \begin{bmatrix} \textsc{cat}: & ap \\ \textsc{t}: & X \end{bmatrix} \begin{bmatrix} \textsc{cat}: & bp \\ \textsc{t}: & X \end{bmatrix} \begin{bmatrix} \textsc{cat}: & cp \\ \textsc{t}: & X \end{bmatrix}$$

Example: (continued)

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

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

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

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

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

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

Example: (continued)

$$\left[\text{CAT}: \quad \mathit{at}\right] \quad \longrightarrow \quad a$$

$$\left[\text{CAT}: \quad \mathit{bt}\right] \quad \longrightarrow \quad b$$

$$\left[\text{CAT}: \quad \mathit{ct}\right] \quad \longrightarrow \quad c$$

Trans-context-free languages

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

Start with a form that consists of the start symbol,

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

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

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

Example: (continued)

Applying ρ_2, to the first element of σ_1:

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

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

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

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

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

Example: (continued)

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

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

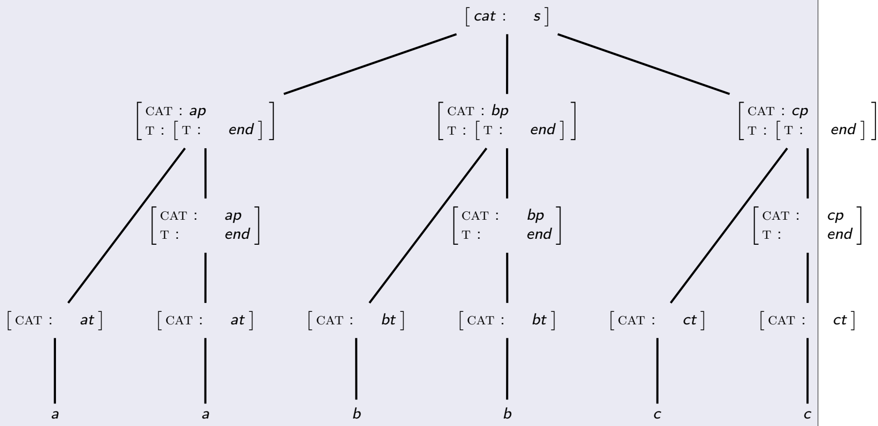
In the same way we can now apply ρ_5 and ρ_7 and obtain, eventually,

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

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

Trans-context-free languages

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



The repetition language

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

The signature of the grammar consists in the features $\textsc{cat}$, $\textsc{first}$ and $\textsc{rest}$ and the atoms *s*, *ap*, *bp*, *at*, *bt* and *elist*. The terminal symbols are *a* and *b*. The start symbol is the left-hand side of the first rule.

Example: (continued)

$$\begin{bmatrix}\textsc{cat}: & s\end{bmatrix} \rightarrow \begin{bmatrix}\textsc{first}: & X\\ \textsc{rest}: & Y\end{bmatrix} \begin{bmatrix}\textsc{first}: & X\\ \textsc{rest}: & Y\end{bmatrix}$$

$$\begin{bmatrix}\textsc{first}: & ap\\ \textsc{rest}: & \begin{bmatrix}\textsc{first}: & X\\ \textsc{rest}: & Y\end{bmatrix}\end{bmatrix} \rightarrow \begin{bmatrix}\textsc{cat}: & at\end{bmatrix} \begin{bmatrix}\textsc{first}: & X\\ \textsc{rest}: & Y\end{bmatrix}$$

$$\begin{bmatrix}\textsc{first}: & bp\\ \textsc{rest}: & \begin{bmatrix}\textsc{first}: & X\\ \textsc{rest}: & Y\end{bmatrix}\end{bmatrix} \rightarrow \begin{bmatrix}\textsc{cat}: & bt\end{bmatrix} \begin{bmatrix}\textsc{first}: & X\\ \textsc{rest}: & Y\end{bmatrix}$$

$$\begin{bmatrix}\textsc{first}: & ap\\ \textsc{rest}: & elist\end{bmatrix} \rightarrow \begin{bmatrix}\textsc{cat}: & at\end{bmatrix}$$

$$\begin{bmatrix}\textsc{first}: & bp\\ \textsc{rest}: & elist\end{bmatrix} \rightarrow \begin{bmatrix}\textsc{cat}: & bt\end{bmatrix}$$

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

Unification grammars and Turing machines

- Unification grammars can simulate the operation of Turing machines.
- The membership problem for unification grammars is as hard as the halting problem.

Unification grammars and Turing machines

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

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

Unification grammars and Turing machines

- A **configuration** of a Turing machine consists of the state, the contents of the tape and the position of the head on the tape.
- A configuration is depicted as a quadruple (q, w_l, σ, w_r) where $q \in Q$, $w_l, w_r \in \Sigma^*$ and $\sigma \in \Sigma$; in this case, the contents of the tape is $\flat^\omega \cdot w_l \cdot \sigma \cdot w_r \cdot \flat^\omega$, and the head is positioned on the σ symbol.
- A given configuration yields a *next configuration*, determined by the transition function δ, the current state and the character on the tape that the head points to.

Unification grammars and Turing machines

Let

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

Unification grammars and Turing machines

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

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

Unification grammars and Turing machines

- A next configuration is only defined for configurations in which the state is not the final state, h.
- Since δ is a total function, there always exists a unique next configuration for every given configuration.
- We say that a configuration c_1 *yields* the configuration c_2, denoted $c_1 \vdash c_2$, iff c_2 is the next configuration of c_1.

Unification grammars and Turing machines

Program:

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

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

- if there were a decision procedure to determine whether $w \in L(G)$ for an *arbitrary* unification grammar G, then in particular such a procedure could determine membership in the language of G_M, simulating the Turing machine M.
- the procedure for deciding whether $w \in L(G)$, when applied to the problem halt$\in L(G_M)$, determines whether M terminates for the empty input, which is known to be undecidable.

Unification grammars and Turing machines

- Feature structures will have three features: *curr*, representing the character under the head; *right*, representing the tape contents to the right of the head (as a list); and *left*, representing the tape contents to the left of the head, in a reversed order.
- All the rules in the grammar are unit rules; and the only terminal symbol is halt. Therefore, the language generated by the grammar is necessarily either the singleton {halt} or the empty set.

Unification grammars and Turing machines: signature

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

- $\textsc{Feats} = \{\textsc{cat},\ \textsc{left},\ \textsc{right},\ \textsc{curr},\ \textsc{first},\ \textsc{rest}\}$
- $\textsc{Atoms} = \Sigma \cup \{\mathit{start},\ \mathit{elist}\}$.
- The start symbol is $\left[\textsc{cat}:\ \ \mathit{start}\right]$.
- the only terminal symbol is halt.

Unification grammars and Turing machines: rules

Two rules are defined for every Turing machine:

$$\begin{array}{lcl}
\begin{bmatrix}\textsc{cat}: & \mathit{start}\end{bmatrix} & \rightarrow & \begin{bmatrix}\textsc{cat}: & s \\ \textsc{curr}: & \flat \\ \textsc{right}: & \mathit{elist} \\ \textsc{left}: & \mathit{elist}\end{bmatrix} \\
h & \rightarrow & \text{halt}
\end{array}$$

Unification grammars and Turing machines: rules

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

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

Unification grammars and Turing machines: rules

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

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

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

Unification grammars and Turing machines: rules

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

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

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

Unification grammars and Turing machines: results

Lemma

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

Theorem

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

Corollary

The universal recognition problem for unification grammars is undecidable.