Chart parsing with non-atomic categories

L545
Spring 2010

With thanks to Detmar Meurers
The issue

• Parsing strategies and memoization (well-formed substring tables, charts) discussed with atomic categories.
  – Example: $S \rightarrow NP \ VP$

• How about the compound terms used as categories?
  – Example: $S \rightarrow NP(Per,Num) \ VP(Per,Num)$
Overview

Three options for parsing with grammars using non-atomic categories:

1. Expand the grammar into a CFG with atomic categories

2. Parse using an atomic CFG backbone with reduced information

3. Incorporate special mechanisms into the parser
Idea 1: Transform into CFG with atomic categories

If only compound terms without variables are used as categories, the rules directly correspond to rules with atomic categories.

Example:

- $S \rightarrow NP(1,sg) \ VP(1,sg)$

- $S \rightarrow NP_{1sg} \ VP_{1sg}$
More on Idea 1

If there are a finite set of possible values for the variables occurring in the compound terms, it is possible to replace a rule with the instances for all possible instantiations of variables.

Example:

• $S \rightarrow NP(\text{Per,Num}) \ VP(\text{Per,Num})$

• $S \rightarrow NP(1,\text{sg}) \ VP(1,\text{sg})$
  $S \rightarrow NP(2,\text{sg}) \ VP(2,\text{sg})$
  $S \rightarrow NP(3,\text{sg}) \ VP(3,\text{sg})$
  $S \rightarrow NP(1,\text{pl}) \ VP(1,\text{pl})$
  $S \rightarrow NP(2,\text{pl}) \ VP(2,\text{pl})$
  $S \rightarrow NP(3,\text{pl}) \ VP(3,\text{pl})$
Evaluation of Idea 1

- leads to a potentially huge set of rules (number of categories grows exponentially w.r.t. the number of features)
  - grammar size relevant for time and space efficiency of parsing
- doesn’t allow for variables, i.e., misses generalizations
Idea 2: Parse using atomic CFG backbone (reduced info)

• idea:
  – parse using a property defined for all categories
  – use other properties to filter solutions from set of parses

• downside:
  – parsing with partial information can significantly enlarge the search space
Idea 3: Incorporate special mechanism into parser

- How two categories are combined has to be replaced by unification.

- Every active and inactive edge in a chart may be used for different uses. So for each time an edge is used, a new copy needs to be made.

- Two effectiveness issues:
  - Use subsumption test to ensure general enough predictions
  - Using restriction to prevent prediction loops

- Two efficiency issues (not dealt with here):
  - intelligent indexing of edges in chart
  - packing of similar edges in chart (cf., Tomita parser)
Where we’re going

First, we need to thoroughly explore using non-atomic categories

- Feature Structures and Unification
- Unification-Based Grammars
- Chart Parsing with Unification-Based Grammars
- Type Hierarchies
Feature structures

- To address the problem of adding agreement to CFGs, we need features, e.g., a way to say:

\[
\begin{bmatrix}
\text{NUMBER} & \text{sg} \\
\text{PERSON} & 3
\end{bmatrix}
\]

- A structure like this allows us to state properties, e.g., about a noun phrase

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{NUMBER} & \text{sg} \\
\text{PERSON} & 3
\end{bmatrix}
\]

- Each feature (e.g., NUMBER) is paired with a value (e.g., sg)
  - A bundle of feature-value pairs can be put into an attribute-value matrix (AVM)
Constraints

What we’re doing is saying that each rule of the grammar is a complex bundle of constraints

- $S \rightarrow NP \ VP$ means that an $S$ object is constrained to be composed of an $NP$ followed by a $VP$

Features allow us to add more constraints

- $S \rightarrow NP \ VP$ only if the number of $NP = \text{the number of } VP$
  - Constraint 1: $S \rightarrow NP \ VP$
  - Constraint 2: $NP \ NUM = VP \ NUM$

So, what we are delving into is constraint-based processing
Feature paths

Values can be atomic (e.g. \( sg \) or \( NP \) or \( 3 \)):

\[
\begin{bmatrix}
\text{NUMBER} & sg \\
\text{PERSON} & 3
\end{bmatrix}
\]

Or they can be complex, allowing for feature paths:

\[
\begin{bmatrix}
\text{CAT} & NP \\
\text{AGREEMENT} & \begin{bmatrix}
\text{NUMBER} & sg \\
\text{PERSON} & 3
\end{bmatrix}
\end{bmatrix}
\]

The value of the path \([\text{AGREEMENT}|\text{NUMBER}]\) is \( sg \)

- Complex values allow for more expressivity than a CFG, i.e., can represent more linguistic phenomena
Feature structures as graphs

- Technically, feature structures are directed acyclic graphs (DAGs)
- The feature structure represented by the attribute-value matrix (AVM):

\[
\begin{bmatrix}
\text{CAT} & \text{NP} \\
\text{AGR} & \begin{bmatrix} \text{NUM} & \text{sg} \\ \text{PER} & 3 \end{bmatrix}
\end{bmatrix}
\]

is really the graph:
Reentrancy (structure sharing)

Feature structures embedded in feature structures can share the same values

- Two features share precisely the same object as their value
  - Well indicate this with a tag like $\mathbb{1}$

\[
\begin{align*}
\text{CAT} & \quad S \\
\text{HEAD} & \quad \text{AGR} \quad \mathbb{1} \\
& \quad \text{SUBJ} \quad \text{AGR} \quad \mathbb{1}
\end{align*}
\]

- In this example, the agreement features of both the matrix sentence and embedded subject are identical (same object)
  - This is referred to as **reentrancy**
What structure-sharing is not

- This is structure-sharing (changing value in one place changes both):

\[
\begin{bmatrix}
\text{HEAD} \\
\text{AGR} \\
\text{SUBJ}
\end{bmatrix}
\begin{bmatrix}
\text{NUM} \\
\text{sg} \\
\text{PER} \\
3
\end{bmatrix}
\]

- This is not (changing one value doesn’t change other):

\[
\begin{bmatrix}
\text{HEAD} \\
\text{AGR} \\
\text{SUBJ}
\end{bmatrix}
\begin{bmatrix}
\text{NUM} \\
\text{sg} \\
\text{PER} \\
3
\end{bmatrix}
\]

Drawing out the DAGs for these can help show this
**Unification**

Well often want to merge feature structures

- **Unification** ($\sqcup$) = a basic operation to merge two feature structures into a resultant feature structure (FS)

The two feature structures must be compatible, i.e., have no values that conflict

- Identical FSs: $\left[\text{NUMBER } sg \right] \sqcup \left[\text{NUMBER } sg \right] = \left[\text{NUMBER } sg \right]$

- Conflicting FSs: $\left[\text{NUMBER } sg \right] \sqcup \left[\text{NUMBER } pl \right] = \text{Fail}$

- Merging with an unspecified FS: $\left[\text{NUMBER } sg \right] \sqcup [] = \left[\text{NUMBER } sg \right]$
Unification (cont.)

• Merging FSs with different features specified:
  
  \[
  \begin{bmatrix}
    \text{NUMBER} & \text{sg} \\
    \text{PERSON} & 3
  \end{bmatrix}
  \sqcup
  \begin{bmatrix}
    \text{NUMBER} & \text{sg} \\
    \text{PERSON} & 3
  \end{bmatrix}
  =
  \begin{bmatrix}
    \text{NUMBER} & \text{sg} \\
    \text{PERSON} & 3
  \end{bmatrix}
  \]

• More examples:
  
  \[
  \begin{bmatrix}
    \text{CAT} & \text{NP} \\
    \text{AGR} & \begin{bmatrix}
      \text{NUM} & \text{sg}
    \end{bmatrix}
  \end{bmatrix}
  \sqcup
  \begin{bmatrix}
    \text{CAT} & \text{NP} \\
    \text{AGR} & \begin{bmatrix}
      \text{NUM} & \text{sg}
    \end{bmatrix}
  \end{bmatrix}
  =
  \begin{bmatrix}
    \text{CAT} & \text{NP} \\
    \text{AGR} & \begin{bmatrix}
      \text{NUM} & \text{sg}
    \end{bmatrix}
  \end{bmatrix}
  \]

  \[
  \begin{bmatrix}
    \text{AGR} & \begin{bmatrix}
      \text{NUM} & \text{sg}
    \end{bmatrix} \\
    \text{SUBJ} & \begin{bmatrix}
      \text{AGR} & \begin{bmatrix}
        \text{NUM} & \text{sg}
      \end{bmatrix}
    \end{bmatrix}
  \end{bmatrix}
  \sqcup
  \begin{bmatrix}
    \text{SUBJ} & \begin{bmatrix}
      \text{AGR} & \begin{bmatrix}
        \text{NUM} & \text{sg}
      \end{bmatrix}
    \end{bmatrix}
  \end{bmatrix}
  =
  \begin{bmatrix}
    \text{AGR} & \begin{bmatrix}
      \text{NUM} & \text{sg}
    \end{bmatrix} \\
    \text{SUBJ} & \begin{bmatrix}
      \text{AGR} & \begin{bmatrix}
        \text{NUM} & \text{sg}
      \end{bmatrix}
    \end{bmatrix}
  \end{bmatrix}
  \]

  \[
  \begin{bmatrix}
    \text{AGR} & \begin{bmatrix}
      \text{NUM} & \text{sg}
    \end{bmatrix} \\
  \end{bmatrix}
  \sqcup
  \begin{bmatrix}
    \text{AGR} & \begin{bmatrix}
      \text{NUM} & \text{sg}
    \end{bmatrix}
  \end{bmatrix}
  =
  \begin{bmatrix}
    \text{AGR} & \begin{bmatrix}
      \text{NUM} & \text{sg}
    \end{bmatrix}
  \end{bmatrix}
  \]
Unification with Reentrancies

- Remember that structure-sharing means they are the same object:

\[
\begin{bmatrix}
\text{AGR} & [\text{NUM } sg] \\
\text{SUBJ} & [\text{AGR } 1]
\end{bmatrix} \sqcup \begin{bmatrix}
\text{SUBJ} & [\text{AGR } \begin{bmatrix} \text{PER } 3 \end{bmatrix}] \\
\end{bmatrix} = \begin{bmatrix}
\text{AGR} & [\text{NUM } sg] \\
\text{SUBJ} & [\text{AGR } 1]
\end{bmatrix}
\]

- When unification takes place, shared values are copied over:

\[
\begin{bmatrix}
\text{AGR} & [\text{PER } 3] \\
\text{SUBJ} & [\text{AGR } 1]
\end{bmatrix} \sqcup \begin{bmatrix}
\text{SUBJ} & [\text{AGR } \begin{bmatrix} \text{NUM } sg \end{bmatrix}] \\
\end{bmatrix} = \begin{bmatrix}
\text{AGR} & [\text{PER } 3] \\
\text{SUBJ} & [\text{AGR } \begin{bmatrix} \text{NUM } sg \end{bmatrix}]
\end{bmatrix}
\]
Unification with Reentrancies (cont.)

• And remember that having similar values is not the same as structure-sharing:

\[
\begin{align*}
\begin{bmatrix}
\text{AGR} & \begin{bmatrix} \text{NUM} & \text{sg} \end{bmatrix} \\
\text{SUBJ} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{NUM} & sg \end{bmatrix} \end{bmatrix}
\end{bmatrix}
\quad \sqsubseteq \\
\begin{bmatrix}
\text{SUBJ} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{PER} & 3 \end{bmatrix} \end{bmatrix}
\end{bmatrix}
\end{align*}
\]

\[
= 
\begin{bmatrix}
\text{AGR} & \begin{bmatrix} \text{NUM} & \text{sg} \end{bmatrix} \\
\text{SUBJ} & \begin{bmatrix} \text{AGR} & \begin{bmatrix} \text{PER} & 3 \end{bmatrix} \end{bmatrix}
\end{bmatrix}
\]
Subsumption

We can say that a more general feature structure (less values specified) subsumes a more specific feature structure

(1) $[\text{NUM} \ sg]$  
(2) $[\text{PER} \ 3]$  
(3) $\begin{bmatrix} \text{NUM} & sg \\ \text{PER} & 3 \end{bmatrix}$

So, we have the following subsumption relations, where

- (1) subsumes (3)  
- (2) subsumes (3)  
- (1) does not subsume (2), and (2) does not subsume (1)
Implementing Unification

How do we implement a check on unification?

• i.e., given feature structures $F_1$ and $F_2$, return $F$, the unification of $F_1$ and $F_2$

Unification is a recursive operation:

• If a feature has an atomic value, see if the other FS has that feature with the same value
  – $[F \ a]$ unifies with $[], [F]$, and $[F \ a]$

• If a feature has a complex value, follow the paths to see if they’re compatible and have the same values at bottom
  – Does $[F \ G_1]$ unify with $[F \ G_2]$? We have to inspect $G_1$ and $G_2$ to find out.

• To avoid cycles, do an **occur check** to see if we’ve seen a FS before or not
The need for unification

Let’s say that we hypothesize that we have:

- a verb which selects for a 3rd person singular noun subject
- a subject which is 2nd person singular

What the verb specifies for the subject has to be able to unify with what the subject is

- In this case, unification will fail (person feature doesn't unify)
Unification-based grammars (Grammars with feature structures)

One way to encode features is to augment a CFG skeleton with feature structure path equations

- CFG skeleton
  \[ S \rightarrow NP \ VP \]

- Path equations
  \[ (NP \ \text{AGREEMENT}) = (VP \ \text{AGREEMENT}) \]

Conditions:

1. There can be zero or more path equations for each rule skeleton \(\rightarrow\) no longer atomic
2. When a path equation references constituents, they can only be constituents from the CFG rule
Handling Linguistic Phenomena

Well look at 3 different phenomena that feature-based, or unification-based, grammars capture fairly succinctly:

1. Agreement

2. Subcategorization

3. Long-distance dependencies
1) Agreement in Feature-based Grammars

One way to capture agreement rules:

\[ S \rightarrow NP \ VP \]
\[ (S \text{ HEAD}) = (VP \text{ HEAD}) \]
\[ (NP \text{ HEAD AGR}) = (VP \text{ HEAD AGR}) \]
\[ VP \rightarrow V \ NP \]
\[ (VP \text{ HEAD}) = (V \text{ HEAD}) \]
\[ NP \rightarrow D \ Nom(inal) \]
\[ (NP \text{ HEAD}) = (Nom \text{ HEAD}) \]
\[ (Det \text{ HEAD AGR}) = (Nom \text{ HEAD AGR}) \]
\[ Nom \rightarrow \text{Noun} \]
\[ (Nom \text{ HEAD}) = (Noun \text{ HEAD}) \]
\[ Noun \rightarrow \text{flights} \]
\[ (Noun \text{ HEAD AGR NUM}) = pl \]
Percolating Agreement Features

S
  [HEAD 4]
    NP
      [HEAD 3[AGR 1]]
        Det
          [HEAD 1[AGR 1]]
        Nom
          [HEAD 3[AGR 1]]
        Noun
          [HEAD 3[AGR 1[NUM pl]]]
          flights
    VP
      [HEAD 4[AGR 1]]
        V
          NP
            [HEAD 4]
Head features in the grammar

- An important concept shown in the previous rules is that heads of grammar rules share properties with their mothers, e.g.:

  \[
  \text{VP} \rightarrow \text{V NP} \\
  \text{(VP HEAD)} = (\text{V HEAD})
  \]

- Knowing the head will tell you about the whole phrase
  - This is important for many parsing techniques
2) Subcategorization

We could specify subcategorization like so:

\[
\begin{align*}
\text{VP} & \rightarrow \ V \\
& \quad (V \ \text{SUBCAT}) = \text{intrans} \\
\text{VP} & \rightarrow \ V \ \text{NP} \\
& \quad (V \ \text{SUBCAT}) = \text{trans} \\
\text{VP} & \rightarrow \ V \ \text{NP} \\
& \quad (V \ \text{SUBCAT}) = \text{ditrans}
\end{align*}
\]

But values like \textit{intrans} do not correspond to anything that the rules actually look like.

- To make \texttt{SUBCAT} better match the rules, we can make its value a list of a verb’s arguments, e.g. \textless \text{NP}, \text{PP} \textgreater
Subcategorization rules

\[ \text{VP} \rightarrow \text{V NP PP} \]
\[ (\text{VP HEAD}) = (\text{V HEAD}) \]
\[ (\text{V SUBCAT}) = \langle \text{NP, NP, PP} \rangle \]

\[ \text{V} \rightarrow \text{leaves} \]
\[ (\text{V HEAD AGR NUM}) = \text{sg} \]
\[ (\text{V SUBCAT}) = \langle \text{NP, NP, PP} \rangle \]

There is also a longer, more formal way to specify lists:

\[ <\text{NP,PP}> \text{ is equivalent to: } \begin{bmatrix}
\text{FIRST} & \text{NP} \\
\text{REST} & \begin{bmatrix}
\text{FIRST} & \text{PP} \\
\text{REST} & \langle \rangle \\
\text{REST} & \langle \rangle \\
\end{bmatrix}
\end{bmatrix} \]
Subcategorization Example

```
[HEAD 1]
[SUBCAT ⟨4 NP⟩]

[HEAD 4 [AGR [NUM sg]]]
[SUBCAT ⟨4 NP, 2, 3⟩]

[CAT 2]
[CAT 3]
```

```
VP
[
  HEAD 1
  SUBCAT ⟨4 NP⟩
]

V

NP

PP
```
Handling Subcategorization

How do we ensure that an objects subcategorization list corresponds to what we see in the actual tree?

- We need a subcategorization principle

Roughly speaking, as a tree is built, items are checked off of the $\text{SUBCAT}$ list

- The subcat list must be empty at the top of a tree

- In the previous example, if we had used the rule $\text{VP} \rightarrow \text{V NP}$, then we would have been left with $\text{SUBCAT} \ <\text{NP,PP}>$

- Or, if we had some rule like $\text{VP} \rightarrow \text{V NP PP PP}$, then our $\text{SUBCAT}$ list at the top would be $\ <\text{NP}>$, but with a PP left over
3) Long-distance dependencies

Long-distance dependencies are often also called movement phenomena

- **Topicalization:** *John she likes __*. 
- **Wh-questions:** *Who does she like __?*

But we want to capture this without movement instead, well pass features along the tree

- **Bottom:** introduce a trace
- **Middle:** pass the trace
- **Top:** Unify the features of the trace with some real word (e.g., John or Who above)

Well use a **gap** feature for this
Handling long-distance dependencies

TOP (fill gap):

\[ S \rightarrow \text{WH-word Be-copula NP} \]
\[ (\text{NP GAP}) = (\text{WH-word HEAD}) \]

MIDDLE (pass gap):

\[ \text{NP} \rightarrow \text{D Nom} \]
\[ (\text{NP GAP}) = (\text{Nom GAP}) \]
\[ \text{Nom} \rightarrow \text{Nom RelClause} \]
\[ (\text{Nom GAP}) = (\text{RelClause GAP}) \]
\[ \text{RelClause} \rightarrow \text{RelPro NP VP} \]
\[ (\text{RelClause GAP}) = (\text{VP GAP}) \]

BOTTOM (identify gap):

\[ \text{VP} \rightarrow \text{V} \]
\[ (\text{VP GAP}) = (\text{V SUBCAT SECOND}) \]
What is the flight that you have?
What’s going on

- Traces, or gaps, are allowed as items from \texttt{SUBCAT} lists

- When we introduce a trace, we need to make sure it gets checked off the \texttt{SUBCAT} list, so everything works out with the subcat principle

- Alternate way: the \texttt{GAP} value of a mother of a rule is the union of the daughter’s \texttt{GAP} values
  - So, we wouldn’t have to write separate rules for RelClause, Nom, NP, etc.
  - When a \texttt{SUBCAT} list is empty and there is an item which matches something in the \texttt{GAP} set, we can remove it from \texttt{GAP}
Altering a chart parser to handle unification

Our grammar still has a context-free backbone, so we could just parse a sentence with a CFG and use the features to filter out the ungrammatical sentences.

- But by utilizing unification as we parse, we can eliminate parses that won't work in the end.
  - e.g., we'll eliminate NPs that don't match in agreement features with their VPs as we parse, instead of ruling them out later.
Changes to the chart representation

Each state will be extended to include the LHS feature structure (FS), which can get augmented as it goes along

- i.e., Add a feature structure (in DAG form) to each state
  - So, $S \rightarrow \bullet\text{ NP VP, [0,0]}$
  - Becomes $S \rightarrow \bullet\text{ NP VP, [0,0], FS}_S$

The predictor, scanner, and completer have to pass the FS, so all three operations have to be altered
Earley parser with atomic categories

**Prediction:**
for each \(i[A \rightarrow \alpha \bullet_j B \beta]\) in chart
for each \(B \rightarrow \gamma\) in rules
add \(j[B \rightarrow \bullet_j \gamma]\) to chart

**Scanning:**
let \(w_1 \ldots w_j \ldots w_n\) be the input string
for each \(i[A \rightarrow \alpha \bullet_{j-1} w_j \beta]\) in chart
add \(i[A \rightarrow \alpha w_j \bullet_j \beta]\) to chart

**Completion (fundamental rule of chart parsing):**
for each \(i[A \rightarrow \alpha \bullet_k B \beta]\) and \(k[B \rightarrow \gamma \bullet_j]\) in chart
add \(i[A \rightarrow \alpha B \bullet_j \beta]\) to chart
Earley parser with unification

Prediction:

for each \( i[A \rightarrow \alpha \bullet_j B \beta] \) in chart
for each \( B' \rightarrow \gamma \) in rules
add \( j[\sigma(B \rightarrow \bullet_j \gamma)] \) with \( \sigma = \text{mgu}(B, B') \) to chart

Completion (fundamental rule of chart parsing):

for each \( i[A \rightarrow \alpha \bullet_k B \beta] \) and \( k[B' \rightarrow \gamma \bullet_j] \) in chart
add \( i[\sigma(A \rightarrow \alpha B \bullet_j \beta)] \) with \( \sigma = \text{mgu}(B, B') \) to chart
Prediction

Prediction:

for each $i[A \rightarrow \alpha \bullet_j B \beta]$ in chart
for each $B' \rightarrow \gamma$ in rules
add $j[\sigma(B \rightarrow \bullet_j \gamma)]$ with $\sigma = \text{mgu}(B, B')$ to chart

The predictor takes the specification of $B$ (i.e., FS) and finds the most general unifier (mgu) of $B$ with $B'$

• If $B$ and $B'$ do not unify, then the rule for $B'$ is not added to the chart

• Initially (i.e., at position 0), all that happens is that a dotted rule with a FS is added to the chart
Completion

Completion (fundamental rule of chart parsing):

for each \( i[A \rightarrow \alpha \bullet \gamma B \beta] \) and \( k[B' \rightarrow \gamma \bullet j] \) in chart
add \( i[\sigma(A \rightarrow \alpha B \bullet j \beta)] \) with \( \sigma = \text{mgu}(B, B') \) to chart

Again, a step of unification is added.

- \( B \) and \( B' \) must unify in order for the dot to move
- The resulting (more specific) FS is added to the chart
How to use a chart with feature structures

- Use **unification** to combine categories in completion or prediction.

- Each time a rule or an edge is used, a new **copy** is made.

- But how about testing whether an entry already exists in the chart?
  - Currently, we simply check to see whether a state **unifies** with something already in the chart and do not add a new state if it is already there
  - But a more specific or a more general state may already be in the chart
The subsumption problem (based on Covington 1994)

- S → NP VP
- NP → Det N
- VP → V'(0)
- VP → V'(X) Comps(X)
- V'(X) → V(X)
- V'(X) → Adv V(X)
- Comps(1) → NP
- Comps(2) → NP NP

- Det → the
- N → dog
- N → cat
- Adv → often
- V(0) → sings
- V(1) → chases
- V(2) → gives
The subsumption problem (2)

What happens when we try to parse *the dog chases the cat*?

- At position 2 (between *dog* and *chases*), from 2 to 2, the parser predicts:
  - $\text{VP} \rightarrow \bullet \ V'(0)$
  - $V'(0) \rightarrow \bullet V(0)$
  - $V'(0) \rightarrow \bullet \text{Adv} \ V(0)$
  - $\text{VP} \rightarrow \bullet \ V'(X) \text{ Comps}(X)$

- What happens when we scan *chases*?
  - We have a passive $V(1)$ edge
  - But there is no predicted $V'(1)$ edge—only $V'(0)$
Using subsumption to check the chart

Subsumption check: Do not add a state to the chart if an equivalent or more general state is already there.

- So, if we want to add a singular determiner state at \([x, y]\), and the chart already has a determiner state at \([x, y]\) unspecified for number, then we do not add it.

- If we don’t impose a subsumption restriction, we could add two states at \([x, y]\), one expecting to see a singular determiner, the other just a determiner.
  - On seeing a singular determiner, the parser advances the dot on both rules, creating two edges (since singular unifies with singular and with unspecified).
  - As a result, we would get duplicate edges.

- With subsumption, if either a singular or plural determiner is encountered, we advance the dot, creating only one edge (singular or plural) at \([x, y]\)
Checking for subsumption

Case 1

Let’s define a function `subsumes_chk` which takes 2 arguments: more general item & more specific item

No variables:

- `subsumes_chk(V'(1),V'(1)).` → yes
- `subsumes_chk(V'(1),V'(2)).` → no

Compound terms without variables are either identical or different, i.e., here: subsumption = unification
Checking for subsumption
Case 2

Variables only in more general term:
- `subsumes_chk(V'(X),V'(1))`. → yes
- `subsumes_chk(foo(X,X),foo(1,1))`. → yes
- `subsumes_chk(foo(X,X),foo(1,2))`. → no

Succeeds if a consistent variable assignment exists, i.e., here:
subsumption = unification
Checking for subsumption
Case 3

Variables in both terms:
- `subsumes_chk(vbar(X),vbar(Y))`. → yes
- `subsumes_chk(vbar(X),vbar(foo(1,Y)))`. → yes
- `subsumes_chk(vbar(foo(1,2)),vbar(foo(1,Y)))`. → no

- Succeeds if terms can be unified without further instantiating more specific term; in other words:
  - Unification should not require a particular instantiation of a variable in the more specific term.

- Idea: Identify each variable in more specific term with a unique, variable-free term; then subsumption = unification.
The restriction problem

Shieber et al 1995: Grammar accepting $ab^n$ with $N$ being instantiated to the successor representation of $n$.

\[
\begin{align*}
\text{start} & \rightarrow r(0, N) \\
r(X, N) & \rightarrow r(s(X), N) \ b \\
r(N, N) & \rightarrow a
\end{align*}
\]

Prediction step with unification will loop:

\[
\begin{array}{c}
\text{1} & \text{pred } r(0, N) \text{ in 1} \\
\text{2} & \text{pred } r(s(0), N) \text{ in 2} \\
\text{3} & \text{pred } r(s(s(0)), N) \text{ in 3} \\
\text{4} & \text{pred } r(s(s(s(0))), N) \text{ in 3} \\
\vdots
\end{array}
\]

\[
\begin{array}{c}
0[\text{start} \rightarrow \bullet_0 r(0, N)] \\
0[r(0, N) \rightarrow \bullet_0 r(s(0), N) \ b] \\
0[r(s(0), N) \rightarrow \bullet_0 r(s(s(0)), N) \ b] \\
0[r(s(s(0)), N) \rightarrow \bullet_0 r(s(s(s(0))), N) \ b] \\
0[r(s(s(s(0))), N) \rightarrow \bullet_0 r(s(s(s(s(0)))), N) \ b] \\
\vdots
\end{array}
\]
Using restriction to prevent prediction loops

- Prediction terminates for grammars with atomic categories, since a new item is only added to the chart if not already there and there is a finite number of atomic categories.

- Moving beyond atomic categories, there can be an infinite number of non-atomic categories.

- Prediction loop on left-recursive rules can be problem again.

- Solution: restrict number of predicted categories to finitely many cases
Prediction with restriction

for each $i[A \rightarrow \alpha \bullet_j B \beta]$ in chart
  for each $B' \rightarrow \gamma$ in rules
    add $j[\sigma(B \rightarrow \bullet_j \gamma)]$ with $\sigma = \text{restriction}(\text{mgu}(B, B'))$ to chart

$\text{restriction}(\text{mgu}(B, B'))$ can be any operation reducing the number of possible substitutions to finite classes:

- depth bound on term complexity
- elimination of terms that are known to grow indefinitely
- use only of selected terms known not to grow indefinitely

This is sound since prediction only creates a hypothesis to be completed!
Example

Grammar:

\[
\begin{align*}
\text{start} & \rightarrow r(0, N) \\
r(X, N) & \rightarrow r(s(X), N) \ b \\
r(N, N) & \rightarrow a
\end{align*}
\]

Parsing using a restrictor that replaces every term deeper than 2 with a variable:

\[
\begin{align*}
1 & \quad 0[\text{start} \rightarrow \bullet_0 r(0, N)] \\
2 & \quad \text{pred } r(0, N) \text{ in 1} \quad 0[r(0, N) \rightarrow \bullet_0 r(s(0), N) \ b] \\
3 & \quad \text{pred } r(s(0), N) \text{ in 2} \quad 0[r(s(0), N) \rightarrow \bullet_0 r(s(s(0)), N) \ b] \\
4 & \quad \text{pred } r(s(s(A)), N) \text{ in 3} \quad 0[r(s(s(A)), N) \rightarrow \bullet_0 r(s(s(s(A))), N) \ b] \\
5 & \quad \text{pred } r(s(s(A)), N) \text{ in 4} \quad = \text{ edge 4}
\end{align*}
\]
Type hierarchies

Let’s get back to our feature structure formalism ...

• Instead of simple feature structures, formalisms like Head-Driven Phrase Structure Grammar (HPSG) use typed feature structures

• Two problems right now:
  – What prevents us right now from specifying: \([\text{NUM } fem]\)?
  – How can we capture the fact that all values of \(\text{NUM}\) are the same sort of thing, i.e., make a generalization?

Solution: use \textit{types}
Type systems

1. Each feature structure is labeled by a type. \[
\begin{bmatrix}
\text{noun} \\
\text{CASE} & \text{case}
\end{bmatrix}
\]

2. Each type has appropriateness conditions specifying what features are appropriate for it.
   - \textit{noun} \rightarrow [\text{CASE} \ case]
   - \textit{verb} \rightarrow [\text{VFORM} \ vform]

3. Types are organized into a type hierarchy.

4. Unification is modified to allow two different types to unify.
Type hierarchy

If we have specified:

- CASE is appropriate for *noun*
- the value of CASE is *case*
- we have the following type hierarchy:

Then, the following are possible feature structures:

\[
\begin{bmatrix}
\text{noun} \\
\text{CASE nom}
\end{bmatrix} \quad \begin{bmatrix}
\text{noun} \\
\text{CASE acc}
\end{bmatrix} \quad \begin{bmatrix}
\text{noun} \\
\text{CASE dat}
\end{bmatrix}
\]
Unification of types

Now, when we unify feature structures, we have to unify types, too:

- \([\text{CASE} \ case] \sqcup [\text{CASE} \ nom] = [\text{CASE} \ nom]\)

- \([\text{CASE} \ nom] \sqcup [\text{CASE} \ acc] = \text{FAIL}\)

If we specify that acc and dat have a common subtype, obj

- Then, we have the following unification: \([\text{CASE} \ acc] \sqcup [\text{CASE} \ dat] = [\text{CASE} \ obj]\)